Novel Split-Based Approaches to Computing Phylogenetic Diversity and Planar Split Networks

Binh Thanh Nguyen

Supervisor: Prof. Vincent Moulton
Co-supervisor: Dr. Brent Emerson

A thesis submitted for the degree of Doctor of Philosophy
at the University of East Anglia

July 2010

School of Computing Sciences, University of East Anglia
Norwich, United Kingdom

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification at this or any other university or institute of learning.

Acknowledgements

First and foremost, I would like to express my special gratitude to my supervisor, Professor Vincent Moulton, for his invaluable help and support. Not only did he bring me the interest in and courage to work with phylogenetics, for which I had no experience at the time of beginning my research degree, but he also provided me with clear guidance, helpful advice, and patient explanation during the research. His professional working manner, critical comments, and positive thinking made it a great experience working with him and I will carry this with me to the future.

I would like to thank my co-supervisor, Dr. Brent Emerson, who gave me the bridge for understanding the biological needs and applications from computing sciences. The discussions with him about genetic diversity and its relationships to geography were interesting and inspirational.

My work would have been much more difficult without the help from Dr. Andreas Spillner, to whom I would like to express a deep appreciation. His broad knowledge of maths and geometry together with his willingness, carefulness, patience either when discussing, proof reading, or giving comments contributed a lot to the completion of the thesis. His "Germanic" hard-working attitude was also something I learnt from, and that will be helpful for me in the time to come.

My sincere thanks go to Professor Vic Rayward-Smith who supervised my MSc and greatly encouraged me to do a PhD, to Professor Mike Steel (University of Canterbury, New Zealand) for helping me to go to New Zealand and hosting me during the Annual New Zealand Phylogenetics Meeting 2009, and to Dr. Ngọc Nguyễn (Hanoi University of Technology, Vietnam) for his continuous encouragement to study in the last eight years.

I am grateful to friends and colleagues in the Computational Biology group in the School of Computing Sciences at the UEA for their nice and supportive company. I also thank my Vietnamese friends in the UEA and in Vietnam who gave me care and support through the time, which helped maintain the feeling of a strong connection to my home country.

I would like to thank the Engineering and Physical Sciences Research Council for their funding (grant number EP/D068800/1) which allowed me to undertake this research. I am also grateful to the School of Computing Sciences, the UEA and their staff for the nice hospitality in the last five years and for providing me with all facilities needed for my study. I also thank the Ministry of Education and Training of Vietnam and Hanoi University of Technology for providing me with financial and opportunity support for my MSc degree, which opened up the route to the PhD level.

Last but not least, I owe a life-time debt to my family: My father Hòa Nguyễn, my mother Dung Dặng, my sisters Minh Nguyễn and Hằng Nguyễn for their endless encouragement, support, and love.

Publications

A. Spillner, B. Nguyen and V. Moulton. Computing Phylogenetic Diversity for Split Systems. *IEEE/ACM Transactions on Computational Biology and Bioinformatics,* 5:2, 235 - 244, 2008.

B. Nguyen, A. Spillner, B. Emerson, and V. Moulton. Distinguishing between hot-spots and melting-pots of genetic diversity using haplotype connectivity. *Algorithms for Molecular Biology,* 5:19, 2010.

Statement of Originality

Abstract

Phylogenetics is a flourishing research area that studies the evolutionary relationships between species. Phylogenetics has attracted a great deal of attention from researchers in different areas due to its interdisciplinary nature, its importance to understanding evolution, and the increasing availability of molecular sequence data. At the heart of phylogenetics is the reconstruction of evolutionary histories or phylogenetic trees. Based on *splits*, i.e. bipartitions of a set of species, this thesis makes contributions to understanding how to use evolutionary histories to understand the diversity of species, and how to reconstruct more intricate histories for species which evolve in complex ways. Regarding the former, we study the phylogenetic diversity optimisation problem under various situations, prove the NP-hardness for some cases and derive efficient algorithms for the others. Regarding the latter, we develop a new method to reconstruct a planar phylogenetic network from a distance matrix and make an initial investigation of its performance.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Phylogenetics studies the evolutionary relationships among various kinds of organisms, e.g. at the individual, species, or population level, which are nowadays commonly deduced from molecular sequence data. Phylogenetics is interdisciplinary, it makes use of the developments in different fields including Biology, Computing Sciences, Mathematics, and Geometry. Even though it is a young field, phylogenetics has attracted a great deal of attention from researchers, has seen fast moving developments in recent times and has opened up many questions. This thesis focuses on two major aspects of the field: Phylogenetic diversity and phylogenetic networks.

Phylogenetic diversity is a measure used to quantify how evolutionarily diverse a group of taxa is. Although there are a variety of technical definitions of phylogenetic diversity, there is a common problem of finding a group of taxa within a set of given taxa that has the highest diversity. This main problem and its variations, e.g. those with additional constraints on cost for preserving taxa, are important to conservationists who want to understand and/or maintain the diversity of a specific species, or the biodiversity within a geographic region.

Concerning phylogenetic networks, initially, phylogenetic trees were first used to illustrate the evolutionary relationships. Later on it was found that there are evolutionary events such as e.g. hybridization, horizontal gene transfer, and recombination, that cannot be represented with the tree since they are naturally non tree-like. Therefore phylogenetic networks were developed to complement trees. Phylogenetic networks have found practical applications to studying real life data such as mitochondrial DNA or virus genuses.

In this thesis, we develop some novel approaches to phylogenetic diversity and phylogenetic networks based mainly on the mathematics of bipartitions of a set, or splits for short. The contents are structured as follows:

Chapter 2 provides the background information for the thesis. We first introduce some basic concepts in graph theory, then use them to define the key concepts: Phylogenetic trees, collections of splits, or split systems, and split networks. We also briefly describe some common methods used to reconstruct phylogenetic trees and split networks. After that, we place the so-called phylogenetic diversity optimisation problem into the context of split systems (problem MPD, or MaximumPhylogeneticDiversity), which can be specialised to both trees and networks. We also summarise some recent algorithmic developments to the optimisation on trees and networks.

In **Chapter 3**, we study complexity aspects of the MPD problem. Using the reduction technique, which is widely used in studying complexity, we show that not only MPD but many of its variations, either split-based or not, are NP-hard. These results suggest to study the problem for special classes of split systems or using heuristic approaches.

Chapter 4 is dedicated to studying the phylogenetic diversity problem on phylogenetic trees. Inspired by the recently developed greedy approach to the MPD problem on trees, we propose a new algorithm that has the best running time compared to available methods, and is optimal in this respect. We then utilise the special structure of trees to develop a dynamic programming-based, polynomial time algorithm for problem MaximumAverageDiversity which is generally NP-hard.

The MPD problem is studied further in **Chapter 5** under two other special structures of input data: Using some geometric transformation, we translate the optimisation problem on affine split systems to the closely related problem of finding optimal convex polygons, and develop a polynomial time algorithm for this. The geometric approach is then exploited further to study the problem on circular split systems. In particular, we develop an algorithm that has a better running time than the current methods to solve the problem.

Parts of Chapters 3, 4, and 5 are extracted and adapted from the paper "A. Spillner, B. T. Nguyen, and V. Moulton. Computing phylogenetic diversity for split systems. *IEEE/ACM Transactions on Computational Biology and Bioinformatics,* 5(2):235 244, April-June 2008" which was worked out in collaboration with Dr. Andreas Spillner.

Chapter 6 and 7 are dedicated to the improved methods to represent data with split networks. We first introduce flat split systems (FSSs), which are the most general class of split systems whose corresponding split networks can be drawn on the plane, and then give an initial investigation of their application.

In **Chapter 6**, we provide a constructive foundation: We define FSSs, show how to represent them using labelled wiring diagrams, and how to draw their corresponding planar split networks. We then define several basic operations on FSSs and study properties of these operations. These make it possible to manipulate the FSSs and to search through the space of FSSs. In **Chapter 7** we make a comparative study of the application of FSSs: We give a method to find a weighted FSS to represent a given distance matrix and compare this method to the well known NeighborNet method. We evaluate the results on both random and real life data.

We conclude this introduction by noting that in addition to the above, during the course of work undertaken for this thesis, we proposed a method to help analyse the origin of phylogenetic diversity in a species. This resulted in the paper "B. Nguyen, A. Spillner, B. Emerson, and V. Moulton. Distinguishing between hot-spots and melting-pots of genetic diversity using haplotype connectivity. *Algorithms for Molecular Biology,* 5:19, 2010". The work was inspired by the fact that, while phylogenetic diversity is commonly used to measure the diversity at the genetic level, more needs to be understood about how this diversity arises. This is important to conservationists since they need to know if a region with high genetic diversity is a "hot-spot", i.e. the species has come to the region and evolved there for a long time, thus the diversity is stable, or is a "melting-pot", i.e. the high diversity is the result of the different gene pools that have merged in the region only for a short period of time, thus have not become stable yet, and therefore may not be stable enough in the future. We proposed a new measure called *haplotype connectivity* which is easy to understand and efficient to compute given the distance matrix between the taxa. The measure was tested on published data and shown to provide a useful addition to the current tools. We did not include this work in this thesis as it fell somewhat out of the general theme - the computation and interpretation of haplotype connectivity does not require splits at all.

Chapter 2

Background

2.1 Summary

In this chapter, we give some background on phylogenetic trees and phylogenetic networks - the mathematical objects used to represent evolutionary relationships. We first summarise some foundations from graph theory, and then introduce the concept of an X-tree as a bridge between trees in graph theory and phylogenetic trees. After this, phylogenetic trees are defined, and several methods to reconstruct phylogenetic trees are reviewed. We then review split networks - an important special class of phylogenetic networks. We also recall the concept of splits and split systems which are the mathematical objects underlying both X-trees and split networks.

2.2 Phylogenetic trees

Phylogenetic trees are diagrams that are used to depict evolutionary relationships in evolutionary biology. In 1837, in his Notebook B, Darwin sketched what was later considered the first phylogenetic tree ever produced, see a photograph of it in Figure 2.1 (11)[1]. However, Haeckel (71) is usually considered to be the first biologist who published a phylogenetic tree in 1866. There are various forms of

[1]From Wikipedia (`http://en.wikipedia.org/wiki/File:Darwin_tree.png`), URL last checked 30/6/2010. This photograph is in the public domain because the author died prior to 1 January 1940.

phylogenetic trees for various purposes: For example, Haeckel's diagram looks like a real tree (see Figure 2.2), Romer's diagram of vertebrates (125) incorporates an absolute time frame and an estimate of the numbers of species of each group at any one time, Milne and Milne's tree of caddis flies (104) is three dimensional and incorporates factors like habitat and casing construction. To formally define phylogenetic trees, we first recall some related definitions in graph theory.

2.2.1 Basic concepts in graph theory

We now review several concepts in graph theory that will be used later on in this thesis. For more on graphs the reader can refer to e.g. (12; 37). We only consider *undirected graphs*, which we shall refer to simply as graphs.

A *graph* G is a pair (V, E) where V is a non-empty, finite set and E is a subset of $\begin{pmatrix} V \\ 2 \end{pmatrix}$. Here $\begin{pmatrix} M \\ k \end{pmatrix}$ denotes all k-element subsets of a finite set M, $k \in \mathbb{N}$ - the set of natural numbers. The set V is called the set of *vertices* of G and E is called the set of *edges* of G. Sometimes we use $V(G)$ and $E(G)$ to refer to the sets of vertices and edges of a graph G, respectively.

A graph $G = (V, E)$ is called *bipartite* if V can be partitioned into V_1 and V_2 so that for every edge $\{u, v\} \in E$, either $u \in V_1$ and $v \in V_2$ or $u \in V_2$ and $v \in V_1$. A *subgraph* of a graph $G = (V, E)$ is a graph $G' = (V', E')$ with the property that $V' \subseteq V$ and $E' \subseteq E$. Two distinct vertices $u, v \in V$ are called *adjacent* if $\{u, v\} \in E$. The *degree* of a vertex $v \in V$ is the number of vertices that are adjacent to v. An *isomorphism* of graphs $G = (V, E)$ and $G' = (V', E')$ is a bijection $f : V \to V'$ such that, for every $u, v \in V, \{u, v\} \in E$ if and only if $\{f(u), f(v)\} \in E'$. In that case, G and G' are called *isomorphic*.

A *path* from a vertex u to a vertex v in a graph $G = (V, E)$ is a sequence $u = w_0, w_1, w_2, \ldots, w_l = v, l \geq 0$, of pairwise distinct vertices of G such that $\{w_{i-1}, w_i\} \in E$ for all $i \in \{1, 2, \ldots, l\}$. The edges $\{w_{i-1}, w_i\}, i \in \{1, 2, \ldots, l\}$, are called *edges on the path*. The number of edges on the path is called *the length of the path*. A *shortest path* between two vertices u and v of G is a path between u and v that has the minimum length among all paths from u to v. A *cycle* is a path $w_0, w_1, w_2, \ldots, w_l$ such that $l \geq 2$ and $\{w_0, w_l\} \in E$. In a bipartite graph, the number of edges on a cycle is always even.

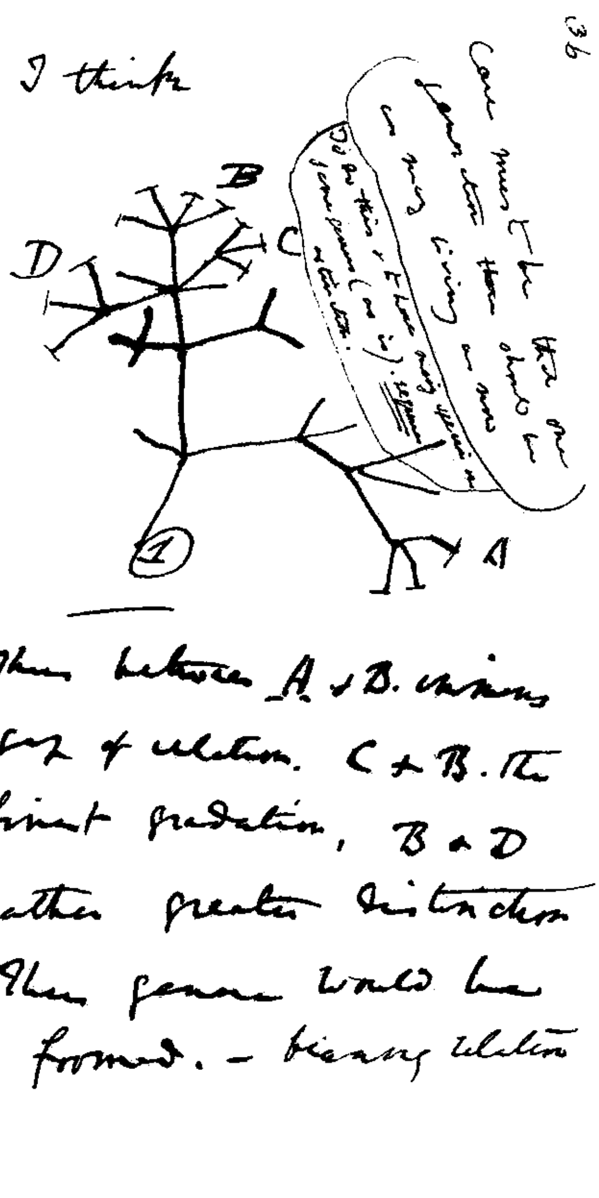

36
I think
Case must be that one generation then should be as many living as now.
To do this & to have many species in same genus (as is) requires extinction.
B
C
D
1
A
Thus between A & B immense gap of relation. C & B the finest gradation, B & D rather greater distinction. Thus genera would be formed. — Bearing relation

Figure 2.1: Darwin's "I think..." tree. The text annotations read: *"I think... Case must be that one generation then should be as many living as now. To do this & to have many species in same genus (as is) requires extinction... Thus between A & B immense gap of relation. C & B the finest gradation, B & D rather greater distinction. Thus genera would be formed. Bearing relation (page 36 ends - page 37 begins) to ancient types with several extinct forms..."*.

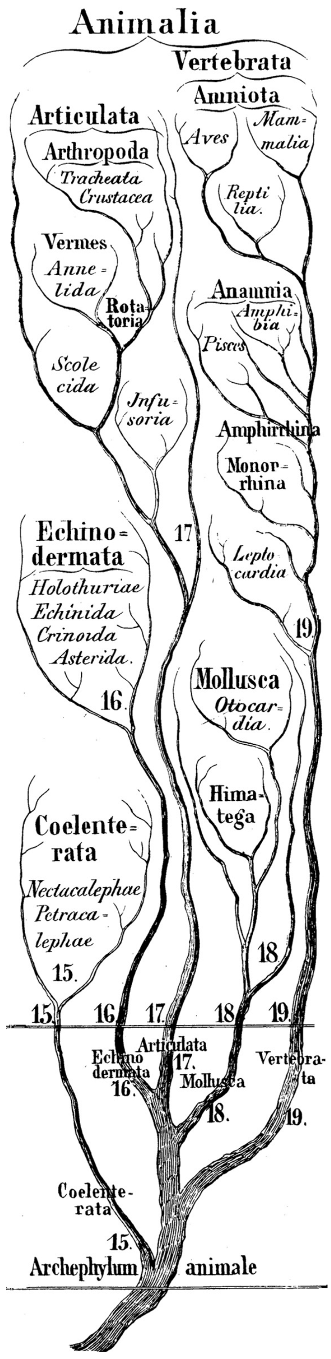


Figure 2.2: Haeckel's phylogenetic tree of the animal kingdom (71).

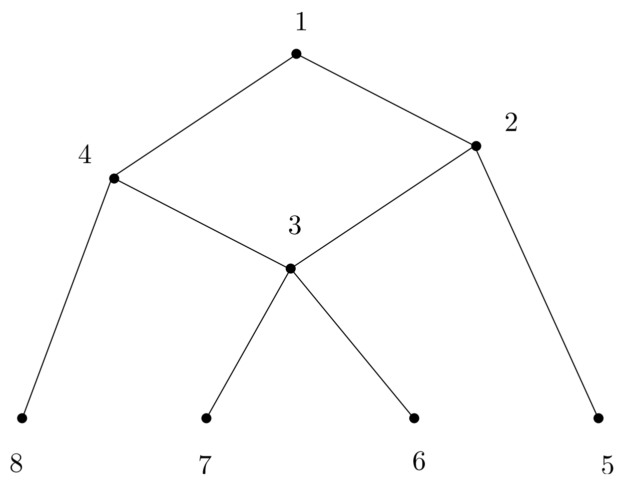


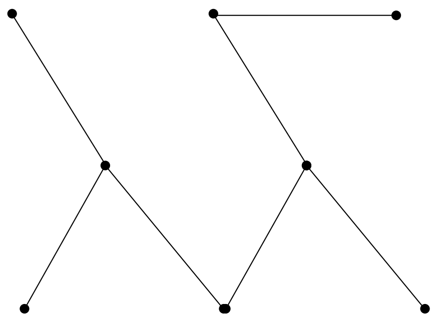
Figure 2.3: An unweighted, connected graph with $V = \{1,2,3,4,5,6,7,8\}$, $E = \{\{1,2\},\{1,4\},\{2,3\},\{3,4\},\{2,5\},\{3,6\},\{3,7\},\{4,8\}\}$. The degree of, for example, vertices 1 and 3 are 2 and 4, respectively; the sequence $5,2,3,4,8$ is a path; the path $1,2,3,4,1$ is a cycle.

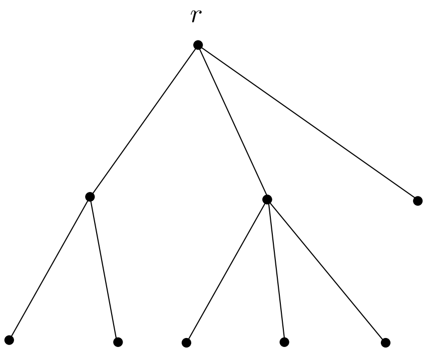
A graph $G = (V, E)$ is *weighted* if there is a function $w : E \to \mathbb{R}_{\geq 0}$ assigning to every edge $e \in E$ a non-negative real number referred to as the *weight* of the edge. On a weighted graph, the length of a path is defined as the total weight of the edges in the path. A graph that is not weighted is called an *unweighted* graph.

A graph $G = (V, E)$ is called *connected* if there is at least one path between any pair of vertices of V. Figure 2.3 shows a connected graph and illustrates some of the terms introduced above.

A *tree* is a connected graph $T = (V, E)$ with no cycles, see Figure 2.4(a). Note that for any pair of vertices u and v of V, there is always a unique path between them. On a tree, a vertex of degree at most 1 is called a *leaf*, a vertex that is not a leaf is called an *inner vertex*. A *subtree* of T is a connected subgraph of T.

A *rooted* tree is a tree $T = (V, E)$ where a special vertex $r \in V$ is distinguished. This vertex introduces a parent-child order on the vertices: Vertex u is the parent of vertex v (or v is the child of u) if u is the predecessor of v on the path from r to v. Figure 2.4(b) depicts a tree rooted at vertex r. The distinguished vertex r is called the *root* of the tree. On a rooted tree $T = (V, E)$, for any $v \in V$,


(a) A tree



(b) A rooted tree

Figure 2.4: Trees in graph theory.

the *subtree rooted at* v is the subtree composed of v, its successors and all edges connecting those vertices.

A rooted tree is called a *binary tree* if the root has degree two and all other inner vertices have degree three. In other words, in a binary tree, every inner vertex has two children. For convenience, two children of an inner vertex of a binary are called *left* and *right* children where "left" and "right" are topologically recognised where the tree is drawn. The subtree rooted at the left and right children of a vertex v are called the *left subtree* and *right subtree* of v, respectively.

For a graph $G = (V, E)$ and a function $f : V \to C$ from V into a non empty, finite set C, the *changing set* of f is the subset

$$Ch(f) = \{\{u, v\} \in E : f(u) \neq f(v)\}$$

of E. The cardinality of $Ch(f)$, denoted by $ch(f)$, is called the *changing number* of f.

2.2.2 Phylogenetic trees.

We now give a formal definition for a phylogenetic trees. An in-depth introduction to such trees may be found in (140).

Let X be a non-empty, finite set. An *X-tree* $\mathfrak{T}$ is an ordered pair (T, ϕ), where $T = (V, E)$ is a tree and $\phi : X \to V$ is a map such that every vertex of degree at most two is contained in $\phi(X)$. An X-tree is also called a *semi-labelled tree* *(on X)*, T is called the *underlying tree* (of $\mathfrak{T}$), and ϕ is called the *labelling map* of $\mathfrak{T}$. Sometimes, when we mention the vertices and the edges of an X-tree, we are actually referring to the vertices and edges of the underlying tree.

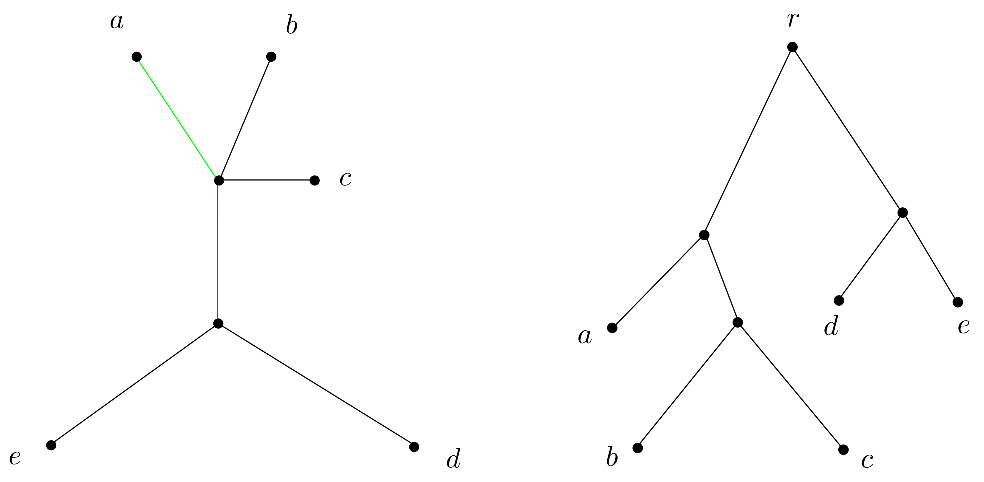
Two X-trees $\mathfrak{T}_1 = (T_1, \phi_1)$ and $\mathfrak{T}_2 = (T_2, \phi_2)$ where $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ are *isomorphic* if there exists a bijection $\psi : V_1 \to V_2$ that is an isomorphism of the graphs T_1 and T_2, and satisfies $\phi_2(x) = \psi(\phi_1(x))$ for all $x \in X$. If ψ exists, it is also unique.

A *phylogenetic X-tree* $\mathfrak{T}$ is an X-tree (T, ϕ) with the property that ϕ is a bijection from X to the set of leaves of T. In other words, a *phylogenetic X-tree* is an X-tree where the leaves are in one-to-one correspondence to the elements of X. A phylogenetic X-tree is also called a *phylogenetic tree* (on X). If every inner vertex of the underlying tree of a phylogenetic X-tree $\mathfrak{T}$ has degree three, $\mathfrak{T}$ is called a *binary phylogenetic tree (on X)*. An example of a phylogenetic tree is pictured in Figure 2.5(a).

An X-tree, and therefore, a phylogenetic tree can be *rooted*. A rooted X-tree $\mathfrak{T}$ is an ordered pair (T, ϕ), where $T = (V, E)$ is a rooted tree with root vertex r, and $\phi : X \to V$ is a map such that for every $v \in V - \{r\}$ of degree at most two, $v \in \phi(X)$. A rooted X-tree is also called a *rooted semi-labelled tree (on X)*.

A rooted phylogenetic X-tree $\mathfrak{T}$, or a rooted phylogenetic tree (on X), is a rooted X-tree (T, ϕ) such that ϕ is a bijection from X to the set of leaves of T and the root r has degree at least two. An example of a rooted phylogenetic tree is pictured in Figure 2.5(b). If the underlying tree is binary, then $\mathfrak{T}$ is called a *rooted binary phylogenetic tree* (on X). Note that the root is considered to represent the common hypothetical ancestor of the taxa represented by the leaves, and the tree represents how the ancestor evolved to become the current (labelled) taxa.

Rooted phylogenetic trees are useful when there is evidence that the taxa in consideration have a common ancestor. In an (unrooted) phylogenetic tree, the common ancestor assumption is not applied; the tree simply illustrates the evolutionary relationships between the taxa. Unrooted phylogenetic trees are important because it is not always possible to determine which of the vertices is, or should be, the common ancestor. In both rooted and unrooted phylogenetic trees, inner unlabelled vertices are considered to represent intermediate unknown or extinct taxa.



(a) A (unrooted) phylogenetic tree.

(b) A rooted, binary phylogenetic tree.

Figure 2.5: Two phylogenetic trees on the set of taxa $X = \{a, b, c, d, e\}$.

In this thesis, we shall focus on unrooted trees. For more about rooted trees see e.g. (53; 82; 140).

2.2.3 X-trees and split systems.

We now define combinatorial objects called splits and split systems which underly phylogenetic trees and, as we shall see, phylogenetic networks which will be defined in the next section. Given a finite set X, an *X-split*, or a *split* of X is a bipartition of X into two non-empty sets A and B. Such a split is denoted as $A|B$, and, since the order of A and B is arbitrary, we consider $A|B$ and $B|A$ as being the same. The subsets A and B are called the *parts* of the split. A split $A|B$ is *trivial* if $|A| = 1$ or $|B| = 1$, otherwise it is *non-trivial*.

Now, consider an X-tree $\mathfrak{T} = (T, \phi)$ with underlying tree $T = (V, E)$. If an edge $e \in E$ is deleted, T will be divided into two subtrees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$. Let $X_1 = \phi^{-1}(V_1)$ and $X_2 = \phi^{-1}(V_2)$. Then both X_1 and X_2 are non-empty, $X_1 \cap X_2 = \emptyset$ and $X_1 \cup X_2 = X$. These properties imply that X_1 and X_2 form a bipartition of X, or $X_1|X_2$ is an X-split. For example, the deletion of the red edge in Figure 2.5(a) gives rise to the split $\{a, b, c\} \,|\, \{d, e\}$. If the split corresponding to the edge $e \in E$ is denoted by S_e, then the edges of the tree $\mathfrak{T}$

induce a set of splits $\mathfrak{S}_{\mathfrak{T}} = \{S_e : e \in E\}$.

A pair of splits $A_1|B_1$ and $A_2|B_2$ is said to be *compatible* if at least one of the four intersections $A_1 \cap A_2$, $A_1 \cap B_2$, $B_1 \cap A_2$, and $B_1 \cap B_2$ is empty. For example, the two splits corresponding to the red and green edges in Figure 2.5(a) are $\{a, b, c\} \,|\, \{d, e\}$ and $\{a\} \,|\, \{b, c, d, e\}$, respectively; and since $\{d, e\} \cap \{a\} = \emptyset$, this pair of splits is compatible. A *split system* (on X) is a subset of the set of all X-splits. A split system is called *compatible* if every pair of splits in the system is compatible.

The *Splits-Equivalence Theorem* by Buneman (26) characterises those split systems that arise from X-trees, in terms of compatible split systems.

Theorem 1. *(Split-Equivalence Theorem (26)) Let $\mathfrak{S}$ be a collection of X-splits. Then, there is an X-tree $\mathfrak{T}$ such that $\mathfrak{S}_{\mathfrak{T}} = \mathfrak{S}$ if and only if every pair of splits in $\mathfrak{S}$ is compatible. Moreover, if such an X-tree exists, then, up to isomorphism, it is unique.*

For illustration, the split system induced by the X-tree in Figure 2.5(a) is $\mathfrak{S} = \{\{a\} \,|\, \{b, c, d, e\}, \{a, c, d, e\} \,|\, \{b\}, \{a, b, d, e\} \,|\, \{c\}, \{a, b, c\} \,|\, \{d, e\}, \{a, b, c, d\} \,|\, \{e\}, \{a, b, c, e\} \,|\, \{d\}\}$. It is easy to check that $\mathfrak{S}$ is compatible.

If the (underlying tree of a) phylogenetic X-tree is weighted, the weight $w(e)$ of an edge e is assigned as the weight of the corresponding split S_e. The length of the path between two elements x and y in X (also called the *phyletic distance* between x and y) is the total weight of the splits corresponding to the edges on the path between $\phi(x)$ and $\phi(y)$. If $S = X_1|X_2$ is such a split, then note that x and y never belong to the same part of S, i.e. either $x \in X_1$ and $y \in X_2$ or $x \in X_2$ and $y \in X_1$. We shall therefore also say that *the split S separates x and y*.

2.2.4 Reconstructing phylogenetic trees.

Having described what a phylogenetic tree is, we now review how such trees are commonly reconstructed. There are basically two main approaches to reconstruct phylogenetic trees from biological data: Character-based and distance-based. Methods in the first direction, e.g. Maximum Parsimony (MP), Maximum

Likelihood (ML) (52), Bayesian phylogenetic inference (152), build the phylogenetic tree directly from so called *character data*, a concept that will be explained below. Here we briefly describe the MP method (42; 91) as an example.

In biology, "characters" describe attributes of the species being studied and can be, for example, morphological (e.g. wings versus no-wings), behavioral (e.g. active-by-day versus active-by-night), or genetic (e.g. the nucleotide at a particular position in a DNA sequence). The central idea underlying MP is to fit character data to a phylogenetic tree in such a way that the total number of changes of characters hypothesized by the tree is minimum, in accordance to 'Ockham's Razor' principle (7) which states that one should always prefer a simpler explanation (less changes, in this case) over a more complicated one (more changes).

Let X be a set of taxa. Mathematically, a *character on* X is a function $\chi : X' \to C$ where $X' \subseteq X, X' \neq \emptyset$ and C is a non-empty, finite set of *character states*. If $X' = X, \chi$ is called a *full character*, and if $|\chi(X')| = r$, χ is called an *r-state character*. A character χ on X is a *binary character* if χ is a two-state full character. Allowing X' to be a proper subset of X means that a character state may not be defined for some taxa.

Let $\mathfrak{T} = (T, \phi)$ be a phylogenetic tree on X where $T = (V, E)$. An *extension of a character* χ on X to $\mathfrak{T}$ is a function $\bar{\chi} : V \to C$ for which $\bar{\chi} \circ (\phi|X') = \chi$, where $\phi|X'$ denotes the restriction of $\mathfrak{T}$ to X'. The *parsimony score* of χ on $\mathfrak{T}$, denoted by $l(\chi, \mathfrak{T})$, is the minimum changing number $ch(\bar{\chi})$ over all extensions $\bar{\chi}$ of χ to $\mathfrak{T}$. If $\bar{\chi}$ is an extension of χ to $\mathfrak{T}$ such that $ch(\bar{\chi}) = l(\chi, \mathfrak{T})$, then $\bar{\chi}$ is called a *minimum extension* of χ to $\mathfrak{T}$.

Let $\mathfrak{C} = \{\chi_1, \chi_2, \ldots, \chi_k\}$ be a sequence of characters on X. The *parsimony score of* $\mathfrak{C}$ on a phylogenetic X-tree $\mathfrak{T}$, denoted by $l(\mathfrak{C}, \mathfrak{T})$, is the sum of the parsimony scores of the characters in $\mathfrak{C}$ on $\mathfrak{T}$, or:

$$l(\mathfrak{C}, \mathfrak{T}) = \sum_{i=1}^{k} l(\chi_i, \mathfrak{T}).$$

A phylogenetic X-tree $\mathfrak{T}$ that *minimises* $l(\mathfrak{C}, \mathfrak{T})$ among all phylogenetic X-trees is called a *maximum parsimony tree* for $\mathfrak{C}$ and the corresponding value of $l(\mathfrak{C}, \mathfrak{T})$ is denoted by $l(\mathfrak{C})$.

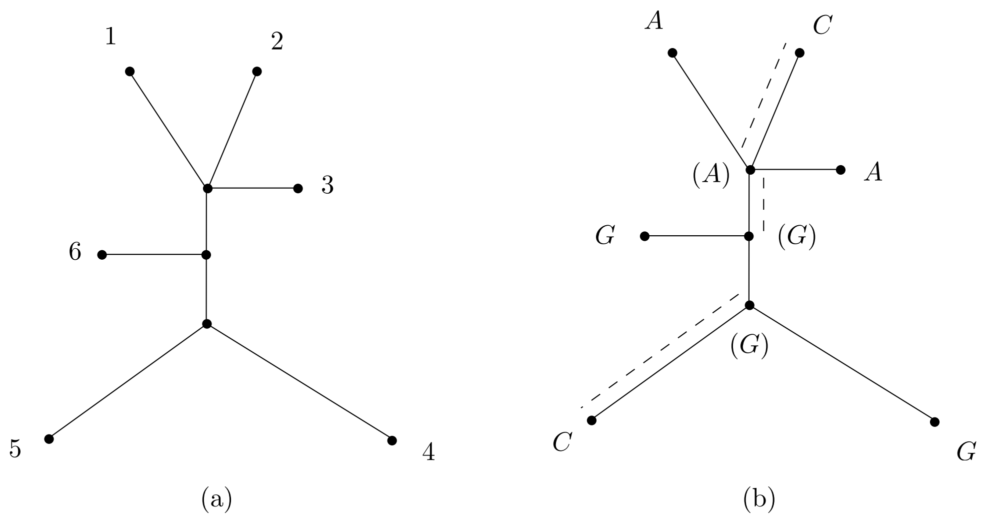


Figure 2.6: A minimum extension of a character.

For example, let $X = \{1, 2, 3, 4, 5, 6\}$ and $\mathfrak{T}$ be the phylogenetic tree on X depicted in Figure 2.6(a). Let $\chi : X \to \{A, C, G\}$ be the character defined by $\chi(1) = \chi(3) = A, \chi(2) = \chi(5) = C$, and $\chi(4) = \chi(6) = G$. It can easily be verified that $l(\chi, \mathfrak{T}) = 3$ with a minimum extension $\bar{\chi}$ of χ to $\mathfrak{T}$ shown in Figure 2.6(b) along with the three corresponding edges in $Ch(\bar{\chi})$ (indicated by extra dashed lines). Note that in general, there can be more than one minimum extension of a character χ on X to a phylogenetic X-tree $\mathfrak{T}$.

If a phylogenetic tree $\mathfrak{T}$ and a full character $\chi : X \to C$ are given, then $l(\chi, \mathfrak{T})$ can be computed quickly either by a general dynamic programming scheme which runs in $O(|X| * |C|^2)$ time (140) or by the *Fitch - Hartigan algorithm* (55; 73) which runs in $O(|X| * |C|)$ time.

However, the problem of finding a maximum parsimony tree for a sequence $\mathfrak{C}$ of binary characters is NP-hard (58). Therefore, different methods have been developed to study the problem for different case. For example, there have been branch-and-bound algorithms for determining optimal and near-optimal maximum parsimony phylogenetic trees from protein sequence data (75). For large data sets, heuristic methods based on tree rearrangements, e.g. Nearest Neighbor Interchange, or NNI (109; 123), Sub-tree Pruning and Regrafting, or SPR, and Tree Bisection and Reconnection, or TBR (6; 53; 140; 142), have been used to find candidate tree(s). These trees will then be labelled using e.g. the Fitch - Hartigan algorithm mentioned above.

The MP method is widely used since it tries to reconstruct the character states for ancestral species/nodes and is relatively robust, i.e. a small change in input data causes a small change in the results and the like. On the negative side, it produces incorrect phylogenetic trees in some cases (53; 91) and is generally slower than distance-based methods, which we will discuss briefly next.

Distance-based methods build a phylogenetic X-tree from a distance matrix d_X on X, i.e. a function $d_X : X \text{ x } X \to \mathbb{R}_{\geq 0}$ where $d_X(v, u) = d_X(u, v)$ and $d_X(u, u) = 0$ for all $u, v \in X$; $d_X(u, v)$ represents the *distance* between u and v.

There are a number of ways to derive a distance matrix from biological data. For example, the Hamming distance (72) between two gene sequences of the same length is the number of positions at which the two nucleotides in the two sequences differ. For other approaches to define a matrix for a set of sequences, the reader can refer to e.g. (66; 105; 113; 134).

There are several methods to reconstruct a phylogenetic tree from a distance matrix, e.g. UPGMA (Unweighted Pair-Group Method Using Arithmetic Mean) (103), Least Squares (56), and Minimum Evolution (126). We now briefly describe the widely-used Neighbor-Joining (NJ) method by Saitou and Nei (127). For explanations concerning the other methods, we refer the reader to (53; 114).

NJ, as well as UPGMA, is an agglomerative algorithm and thus, follows the general agglomerative framework: Starting with a simply structured graph G (depending on the specific algorithm) where each taxon of X is represented by a vertex of G, the method works recursively. At each step of the recursion, a pair of vertices p and q of G is selected based on some criterion and is replaced by a new vertex g. Then p and q are temporarily removed from and g is added into G with suitable adjustments on the edges of the graph. The process continues until only two (or three) vertices remain. At this time those vertices that have been temporarily removed are brought back whilst added vertices are kept and become inner vertices; edges connecting every new inner vertex to the vertices that it replaced are added and the resulting tree is obtained.

NJ starts with a star-like tree, i.e. a phylogenetic X-tree $\mathfrak{T}(T, \phi)$ with $|X|$ edges. Note that up to isomorphism, there is only one such tree. An example of such a tree where $X = \{1, 2, 3, 4, 5, 6, 7, 8\}$ is given in Figure 2.7(a). Let z be the (only) interior vertex of T, L be the set of leaves of T and d be a distance matrix on L where $d(u, v) = d_X(\phi^{-1}(u), \phi^{-1}(v))$ for all $u, v \in L$. The agglomerative process only chooses the pair of vertices to merge from L.

The criterion used to select the pair of vertices to merge is based on a matrix Q which is defined as

$$Q(u, v) = (|L| - 2)d(u, v) - \sum_{w \in L} d(u, w) - \sum_{w \in L} d(v, w)$$

for all $u, v \in L$.

Among the pair(s) of leaves p and q where $Q(p, q)$ is minimum, an arbitrary one is selected and is replaced by a new vertex g. To repeat the agglomerative process, Neighbor-Joining adjusts the tree, adapts d, and puts $L = L \backslash \{p, q\} \cup \{g\}$ so that d becomes the distance matrix on the new set L. To this end, the edges (z, p) and (z, q) are removed from, and the edges (g, p), (g, q) and (z, g) are added to T, see e.g. Figure 2.7(b) where vertices $p = 1, q = 2$ and $g = 9$. Then, the modification of d is made using the following formula:

$$d(w, g) = d(g, w) = \frac{1}{2}\left[d(w, p) - d(p, g)\right] + \frac{1}{2}\left[d(w, q) - d(q, g)\right]$$

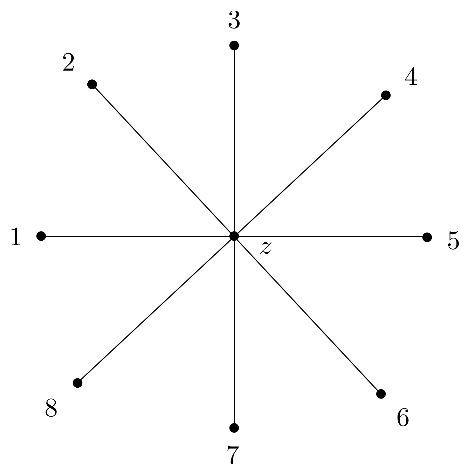
for all $w \in L \backslash \{g\}$. The agglomeration is repeated until L contains only two (when $|X|$=3) or three (when $|X| > 3$) elements.

Not only does Neighbor-Joining incrementally find a phylogenetic X-tree but it also computes edge weights for that tree: Before two vertices p and q are temporarily removed, the weight of the edge $\{p, g\}$ is computed using the following formula:

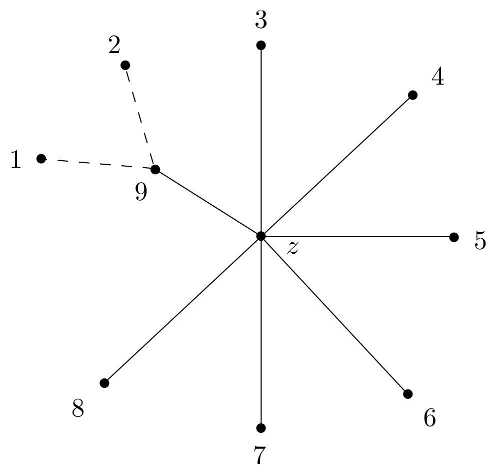
$$d(p, g) = \frac{1}{2}d(p, q) + \frac{1}{2(|L| - 2)}\left[\sum_{w \in L} d(p, w) - \sum_{w \in L} d(q, w)\right],$$

$d(q, g)$ is calculated similarly.

It has been shown that Neighbor-Joining is consistent (8; 21) in the sense that given a weighted phylogenetic X-tree $\mathfrak{T}$, and taking the phyletic distances in $\mathfrak{T}$ between all pairs of elements of X as the input distance matrix, then Neighbor-Joining will reconstruct a phylogenetic X-tree $\mathfrak{T}'$ which is isomorphic to $\mathfrak{T}$.



(a) The star-like starting tree for the Neighbor-Joining algorithm; $L = \{1, 2, 3, 4, 5, 6, 7, 8\}$.



(b) If $Q(1, 2)$ is minimum, then create new node 9 and connect it to 1, 2, and z; $L = \{3, 4, 5, 6, 7, 8, 9\}$.

Figure 2.7: Constructing the Neighbor-Joining tree.

NJ has run time $O(|X|^3)$, hence it can handle reasonably large data sets. On very large data sets, a relaxed version of the algorithm (45; 132) whose running time is $O(|X|^2 \log |X|)$, can be used.

Although being fast, Neighbor-Joining has some disadvantages: It only gives one possible tree and is strongly dependent on the model of evolution used to derive the distance matrix. A model of evolution describes how the process of evolution took place. There are many models of evolution, for example, a simple model by Newman (115), the JC69 model of DNA sequence evolution by Jukes and Cantor (87), causal models (93). However, Neighbor-Joining is still among the most widely used tree reconstruction methods in use (62).

In addition to sequence-based and distance-based approaches, merging phylogenetic trees is also an approach to reconstruct phylogenetic trees. In this approach, given a collection of compatible split systems, one have to find a compatible split system that contains as many splits in the input split systems as possible. Typical methods in this direction are consensus trees (2; 3; 138) and super-trees (13; 14; 128; 131).

Bayesian inference is another approach used to produce phylogenetic trees. The approach assumes a prior probability distribution of the possible trees and works in a way that is closely related to Maximum Likelihood. Bayesian inference methods are usually implemented using Markov chain Monte Carlo sampling algorithms (101; 152).

2.3 Split networks

Phylogenetic trees are not always suitable to represent evolutionary histories since events such as hybridization, horizontal gene transfer and recombination are not tree-like (80). In those cases, phylogenetic networks can be employed.

Phylogenetic networks can be viewed as a generalisation of phylogenetic trees, that represent conflicting signals or alternative evolutionary histories. There are several types of labelled graphs that are generally referred to as phylogenetic networks, as pointed out by Huson and Bryant (84). In particular, they propose to define a phylogenetic network as any network in which taxa are represented by vertices and their evolutionary relationships are represented by edges. This generic concept allows them to classify the diverse usages of the term "phylogenetic networks", see Figure 2.8.

We will focus on split networks and will formally define them later. From now on, by saying "phylogenetic networks", we are referring to "split networks" if not stated otherwise. To learn more about other types of phylogenetic networks, see e.g. (53; 80; 83; 84; 100).

Recall that in Section 2.2, we described how a phylogenetic tree corresponds to a compatible split system. However, not every split system arising from biological data is compatible. The reason is that using (biological) properties of the data set being studied, a collection of splits is derived (usually together with the weight of each split representing the split's importance) without applying any assumption on compatibility. Conflicting and incompatible phylogenetic relationships yield larger classes of split systems, e.g. *weakly compatible* and *circular* split systems as introduced in (9). A split system $\mathfrak{S}$ is called "weakly compatible" if for any three splits $A_1|B_1, A_2|B_2$ and $A_3|B_3$, at least one the the four intersections

$$A_1 \cap A_2 \cap A_3, A_1 \cap B_2 \cap B_3, B_1 \cap A_2 \cap B_3, B_1 \cap B_2 \cap A_3$$

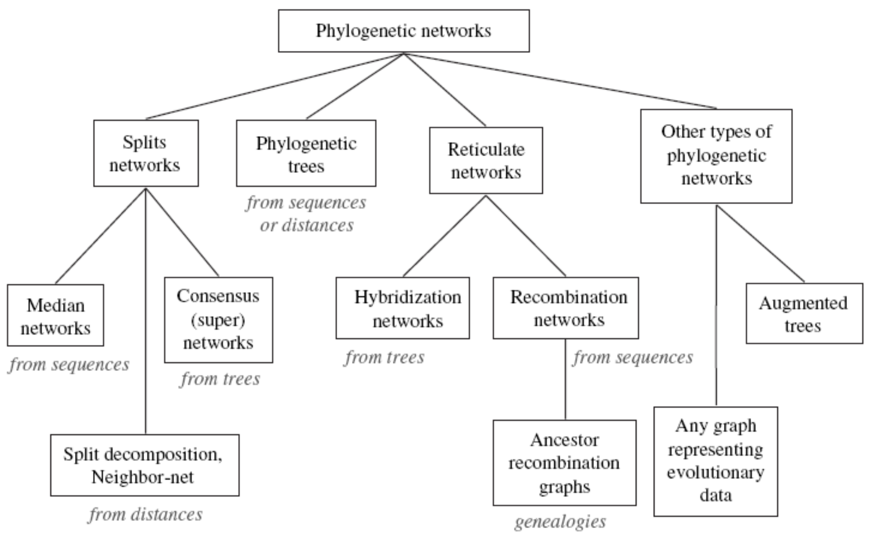


Figure 2.8: An overview of several types of "phylogenetic networks" and how they are related (84). Reused with license (No. 2458771343775) from Oxford University Press.

is empty. Such a split system can be produced by *split decomposition* or related methods (9).

A split system $\mathfrak{S}$ on X is called *circular* if there exists an ordering $\theta = (x_1, x_2, \ldots x_n)$ of the elements of X such that every split $S \in \mathfrak{S}$ has the form $\{x_p, x_{p+1}, \ldots, x_q\}|X\backslash\{x_p, x_{p+1}, \ldots, x_q\}$ for some $p, q \in \mathbb{N}, 1 < p \leq q \leq n$. In this case, we also say that $\theta = (x_1, x_2, \ldots x_n)$ is a *circular order* of X with respect to $\mathfrak{S}$. On a fixed ordering, for two elements $u, v \in X$, we denote by $u < v$ (or $v > u$) that u precedes v in the order, and by $u \leq v$ (or $v \geq u$) that $u < v$ or $u \equiv v$. For every non-empty subset $Y \subseteq X$, the ordering $\theta|Y$ of Y induced by some circular order $\theta = (x_1, x_2, \ldots x_n)$ of X is also defined as a circular order (of Y, with respect to $\mathfrak{S}$). Circular split systems are generated by methods like NeighborNet (23) which will be briefly described later in this section.

Next we formally introduce the concept of split networks, a type of network that is used to represent a split system. To this end, we first define a split graph. Consider a graph $G = (V, E)$, let C be a non-empty, finite set, whose elements are referred to as *colors* and $\sigma : E \to C$ be an *edge coloring*. For a path

$P = w_0, w_1, w_2, ..., w_l$ in G, we define $\sigma(P) = \{\sigma(\{w_{i-1}, w_i\})|i = 1, \ldots, l\}$ and call P *properly colored* if $|\sigma(P)| = l$, that is, the edges on P have pairwise distinct colors. Let $d_G(u, v)$ be the length of a shortest path in G between vertices u and v, and $C_\sigma(u, v)$ be the set of all colors $c \in C$ with the property that c occurs in every shortest path between u and v in G. We call σ an *isometric coloring* if $d_G(u, v) = |C_\sigma(u, v)|$ for all $u, v \in V$, or, equivalently, all shortest paths between any two vertices are properly colored and use the same set of colors. A connected, bipartite graph $G = (V, E)$ is called a *split graph* if there exists an isometric coloring of G.

The next theorem is crucial to understand how split systems are represented by split networks, which will be defined afterward. The proof of the theorem can be found in (39).

Theorem 2. *(39) Let $G = (V, E)$ be a split graph and $\sigma : E \to C$ be an isometric, surjective edge coloring of G. For any $c \in C$, the graph G_c, obtained by deleting all edges of color c, consists of precisely two separate connected subgraphs, which we will denote by $G_c^0 = (V_c^0, E_c^0)$ and $G_c^1 = (V_c^1, E_c^1)$.*

Given a split system $\mathfrak{S}$ on X, a *split network* for $\mathfrak{S}$, denoted by $\mathfrak{N}_{\mathfrak{S}}$, is an ordered pair (G, ϕ), where $G = (V, E)$ is a split graph with the color set $\mathfrak{S}$, and $\phi : X \to V$ is a vertex labelling, such that G together with ϕ *represents* $\mathfrak{S}$ in the sense that every split $S = A|B$ of $\mathfrak{S}$ has the form $S = \cup_{v \in V_S^0} \phi^{-1}(v) | \cup_{v \in V_S^1} \phi^{-1}(v)$, that is, the deletion of all edges of color S produces two connected subgraphs of G, one containing all vertices labelled with elements of A and the other containing all vertices labelled with elements of B.

When it comes to the representation of a split network $\mathfrak{N}_{\mathfrak{S}} = (G, \phi)$, all the edges that are colored by the same split are drawn to be parallel and have the same length to graphically comply with the property of the underlying split graph that all the shortest paths between any two vertices are properly colored and use the same set of colors, see e.g. Figure 2.9.

If $\mathfrak{S}$ is weighted, i.e. there is a weighing function $w : \mathfrak{S} \to \mathbb{R}_{\geq 0}$, the weight of split $S \in \mathfrak{S}$ is assigned to be the length of all the edges colored by S. We now extend the term "phyletic distance" defined for the phylogenetic trees to work on split network G that represents $\mathfrak{S}$: The phyletic distance between two vertices u

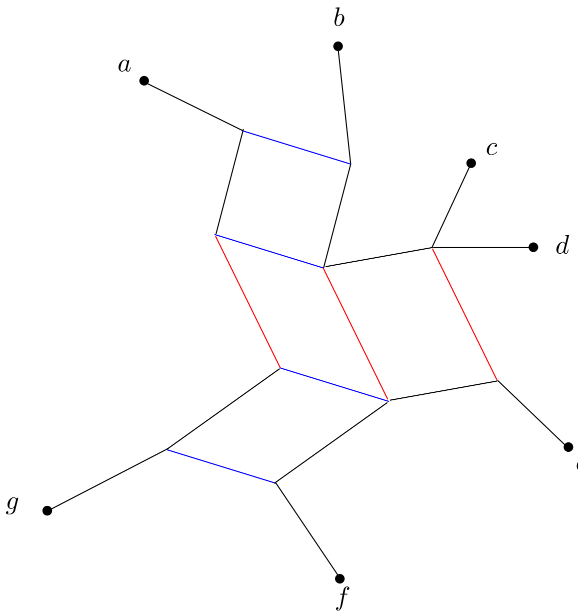


Figure 2.9: An example split network: Red edges represent split $\{a, b, c, d\}|\{e, f, g\}$, blue edges represent split $\{a, g\}|\{b, c, d, e, f\}$.

and v of G, denoted as $d_{\mathfrak{S},w}(u, v)$, is the length of a shortest path between u and v, i.e. the sum of the weights of the splits that separate u and v:

$$d_{\mathfrak{S},w}(u, v) = \sum_{\substack{S=A|B\in\mathfrak{S}, \\ u\in A, v\in B \text{ or} \\ u\in B, v\in A}} w(S). \tag{2.1}$$

If there is no confusion on the weight vector w we will write the function as $d_{\mathfrak{S}}(u, v)$. Equation (2.1) can be used to compute distances between all pairs of vertices to form a distance matrix on X. For this reason, we denote the distance matrix induced by split system $\mathfrak{S}$ with weight vector w as $d_{\mathfrak{S},w}$.

Note that even though a split network represents only one split system, there can be many split networks that represent the same split system as pointed out by Wetzel (149). For this reason, it is not suitable to consider unlabelled nodes in a split network as hypothetical ancestors (9). The main purpose of split networks is to visualise conflicts in the grouping of the taxa supported by the data.

Trying to represent incompatible splits by split networks as outlined above can result in networks that cannot be drawn without any crossing edges, i.e. the networks are non-planar (98). However, a circular split system can always be depicted by a planar split network (111).

There are a variety of methods to reconstruct split networks. Here we briefly describe the *NeighborNet* method which reconstructs a weighted circular split system from a distance matrix. To learn about other methods, the reader is recommended to look at e.g. (80; 81; 85; 100).

NeighborNet uses the same agglomerative approach as Neighbor-Joining. The main difference between the two methods is that in NeighborNet, when two leaves are selected, they are not combined and replaced right away. Instead, the method waits until a leaf is selected a second time, then replaces the three linked leaves by two linked leaves, and applies changes on the distance matrix. Computing the weights is another point where NeighborNet is different from Neighbor-Joining: Neighbor-Joining computes the weights of the branches while the tree is being constructed, using a variant of the least squares formulae. NeighborNet does that after getting all the splits, and uses an optimal least squares with a non-negativity constraint algorithm, which is more costly. Otherwise, NeighborNet chooses the neighboring leaves and adjusts the distance matrix in the same fashion as Neighbor-Joining. For detailed formulae used in NeighborNet, the reader can refer to the original paper (23).

NeighborNet runs in the same time complexity as Neighbor-Joining, i.e. $O(n^3)$, except for computing edge weights. This allows the application of the method on pretty large data sets. Like Neighbor-Joining, NeighborNet is consistent: If the input distance matrix is induced by a weighted circular split system, the method will return the corresponding collection of weighted circular splits, if the input distance matrix is induced by a weighted compatible split system, NeighborNet will return the corresponding weighted compatible split system (23; 25).

NeighborNet only outputs a weighted split system. This split system needs to be drawn. Dress and Huson (39) present algorithms to compute a split network for a given split system, and to draw the network as well. All algorithms mentioned in the paper, including Neighbor-Joining, NeighborNet, Split Decomposition, Parsimony Splits, Consensus Tree, Consensus Network, Super Networks, are included in the software package SplitsTree (84).

2.4 Phylogenetic diversity

In this section, we review some measures of diversity that are based on phylogenetic trees and phylogenetic networks. After the definition of these measures and some related optimisation problems, we will discuss known algorithms to solve those problems.

2.4.1 Phylogenetic diversity on phylogenetic trees

Various authors, e.g. William *et. al.* (150), Nehring and Puppe (112), suggest several diversity measures on a phylogenetic tree and call them all phylogenetic diversities. However, in this research, by “phylogenetic diversity” (PD) we refer to the diversity measure introduced by Faith in 1992 (47). Faith defines the PD of a subset of taxa on a weighted phylogenetic tree as the total weight of the edges on the subtree spanning those taxa. For example, in Figure 2.10, the PD score of taxa $\{b, c, e, f\}$ is the total weight of the **bold** edges and is $2+4+1+3+5+1 = 16$.

Nowadays, there are several theoretical results about computational problems related to PD, and PD is widely accepted and used in practice, see e.g. (10; 46; 48; 49; 50; 57; 110; 124).

A fundamental problem in phylogenetics is to identify subsets of taxa with high diversity. This gives rise to the following optimisation problem: “Given a phylogenetic tree on a set X of n taxa and an integer number k, $1 \leq k \leq n$, find a subset $Y \in X$, $|Y| = k$, so that $PD(Y)$ is maximum among all subsets of X with size k”. Faith proposed a greedy approach to solve this problem: 1) Start with two taxa with maximum pairwise distance. 2) Repeat $(k - 2)$ times: Choose a taxon that contributes most to the current PD score. This algorithm has recently independently been shown to give optimal solutions by Steel (139) and Pardi and Goldman (116). Minh *et. al.* proposed two implementations of the algorithm which run in $O(n \log k)$ and $O(n + (n - k) \log{(n - k)})$, respectively (106).

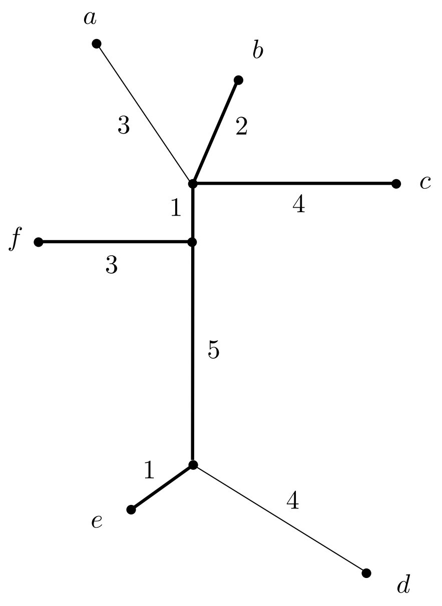


Figure 2.10: A weighted phylogenetic tree, $PD(\{b, c, e, f\}) = 16$.

2.4.2 Phylogenetic diversity on split networks

Consider the compatible split system corresponding to a phylogenetic tree. Note that the PD score of a subset Y of taxa, as defined by Faith, is also the sum of the weights of all splits separating at least two taxa in Y. For example, in Figure 2.10, those splits are $\{a, c, d, e, f\}|\{b\}$, $\{a, b, d, e, f\}|\{c\}$, $\{a, b, c\}|\{d, e, f\}$, $\{a, b, c, d, e\}|\{f\}$, $\{a, b, c, f\}|\{e, d\}$, and $\{a, b, c, d, f\}|\{e\}$.

Hence it is straight forward to generalise PD to general weighted split systems. The PD score of a subset Y of taxa on a split network is the sum of the weights of all splits in the corresponding split system that have at least one taxon of Y in each of the blocks. Mathematically, if we denote by $\mathfrak{S}$ the corresponding split system, and by w the weight function on $\mathfrak{S}$, then:

$$PD_{\mathfrak{S}}(Y) = \sum_{\substack{S=A|B\in\mathfrak{S}\\ A\cap Y\neq\emptyset, B\cap Y\neq\emptyset}} w(S). \tag{2.2}$$

If there is no confusion on the split system in use, we will write $PD_{\mathfrak{S}}(Y)$ as $PD(Y)$ only.

The split-based definition of PD allows the natural extension of the PD optimisation problem on phylogenetic trees to apply on split systems:

MaximumPhylogeneticDiversity (MPD)

Input: a set X with n elements
a split system $\mathfrak{S}$ on X
a split weight function $\omega : \mathfrak{S} \to \mathbb{R}_{>0}$
an integer $k \in \{1, \ldots, n\}$

Output: a subset $Y \subseteq X$ with the property that $|Y| = k$ and
$PD(Y) = \max\{PD(W)|W \subseteq X, |W| = k\}$

This problem will be shown to be NP-hard later in Chapter 3. However, for a circular split system, using the dynamic programming technique, an optimal solution can be computed in polynomial time (107), which we now briefly explain.

In general, dynamic programming is used to solve an optimisation problem that exhibits the characteristics of so-called *overlapping subproblems* and *optimal substructure*. A problem is said to have overlapping subproblems if it can be broken down into subproblems which are reused multiple times. Having the optimal substructure means that the globally optimal solution can be constructed from optimal solutions to subproblems. For more details on dynamic programming, we refer the reader to (33, Chapter 15).

Without loss of generality, it can be assumed that $\theta = (x_1, x_2, \ldots, x_n)$ is a circular order of X with respect to $\mathfrak{S}$, we fix this ordering from now on. The first key observation made in (107) is that for any subset Y of X with ordering $\theta|Y = (y_1, y_2, \ldots, y_k)$, the PD score of Y can be computed using the formula:

$$PD(Y) = \frac{1}{2}\Big(d_{\mathfrak{S}}(y_1, y_k) + \sum_{i=1}^{k-1} d_{\mathfrak{S}}(y_i, y_{i+1})\Big).$$

A collection of k taxa $y_1, y_2, \ldots, y_k$ is called an *ordered k-path* if $x_1 \leq y_1 < y_2 < \cdots < y_k \leq x_n$ with respect to the ordering θ. Its *length* is defined by $L(y_1, y_2, \ldots, y_k) = \sum_{i=1}^{k-1} d_{\mathfrak{S}}(y_i, y_{i+1})$. A *circular k-tour* is achieved from an ordered k-path by adding $y_{k+1} = y_1$ to the list. The tour's length is then $L(y_1, y_2, \ldots, y_k, y_1) = L(y_1, y_2, \ldots, y_k) + d_{\mathfrak{S}}(y_1, y_k)$, which is twice the PD score of $Y = \{y_1, y_2, \ldots, y_k\}$. So finding a subset Y of maximum PD is equivalent to finding a longest circular k-tour. Note that if $(y_1, y_2, \ldots, y_k, y_1)$ is a longest circular k-tour, then $(y_1, y_2, \ldots, y_k)$ is a longest ordered k-path from y_1 to y_k.

This is because the length of such a circular k-tour is only different from that of a k-path by the constant $d_{\mathfrak{S}}(y_k, y_1)$.

For a pair of elements u and v, to find a longest ordered k-path starting at $y_1 = u$ and ending at $y_k = v$, we need to find the $k-2$ intermediate elements that maximise the path's length. Let $y_1, y_2, \ldots, y_k$ be an arbitrary longest ordered k-path from y_1 to y_k, then $y_1, y_2, \ldots, y_{k-1}$ is a longest ordered $(k-1)$-path from y_1 to y_{k-1}. This can be seen as follows: Suppose $y_1, y_2, \ldots, y_{k-2}, y_{k-1}$ is not a longest ordered $(k-1)$-path from y_1 to y_{k-1}. Hence there is an ordered $(k-1)$-path $y_1, y'_2, \ldots, y'_{k-2}, y_{k-1}$ that is longer than $y_1, y_2, \ldots, y_{k-2}, y_{k-1}$. But then $y_1, y'_2, \ldots, y'_{k-2}, y_{k-1}, y_k$ is longer than $y_1, y_2, \ldots, y'_{k-2}, y_{k-1}, y_k$, contradicting that $y_1, y_2, \ldots, y_{k-2}, y_{k-1}$ is a longest ordered k-path from y_1 to y_k, see Figure 2.11. This suggests the following strategy to find the $k-2$ intermediate elements: Consider all possible y_{k-1}. For each y_{k-1}, find a longest ordered $(k-1)$-path from u to y_{k-1}, add up $d_{\mathfrak{S}}(y_{k-1}, v)$ to the length and globally select the maximal value. This reduces the problem of finding a longest ordered k-path to finding a longest ordered $(k-1)$-path and forms the basis for the dynamic programming approach.

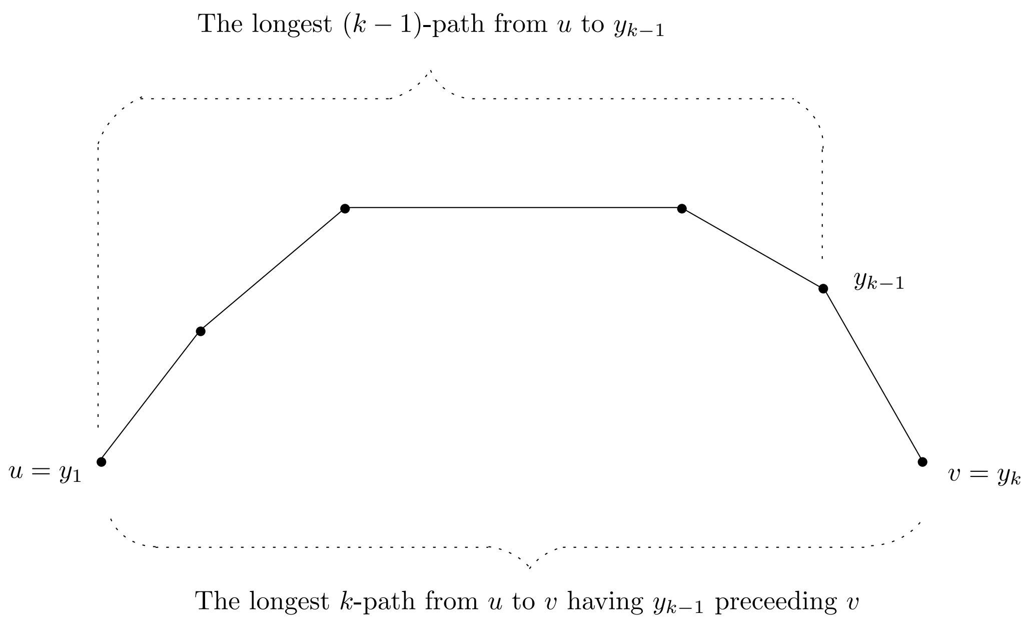

The longest $(k-1)$-path from u to y_{k-1}
y_{k-1}
$u = y_1$
$v = y_k$
The longest k-path from u to v having y_{k-1} preceeding v

Figure 2.11: To find the longest ordered k-path from u to v with a fixed y_{k-1}, one just needs to find the longest ordered $(k-1)$-path from u to y_{k-1}.

We have briefly presented the analysis of the dynamic programming approach with respect to the MPD problem for circular split systems. Consider all pairs of u and v and apply the strategy represented above, one would find the longest ordered k-paths with given endpoints u and v. Among those paths, choose one with the maximum length. As analysed before, the set of taxa on that path is a k-element subset of X with maximum PD.

Formally, for any two taxa u and v, $u < v$, let $L^i(u, v)$ be the length of a longest ordered i-path from u to v and $Y^i(u, v)$ be the set of vertices on that path. Then the length of a longest circular k-tour, denoted as l^k_{max}, can be found by solving the following iterative maximisation:

$$L^i(u,v) = \begin{cases} d_{\mathfrak{S}}(u,v) & \text{if } i = 2, \\ \max_{u<s<v}\{L^{i-1}(u,s) + d_{\mathfrak{S}}(s,v)\} & \text{if } 3 \leq i \leq k, \end{cases} \tag{2.3}$$

$$l^k_{max} = \max_{1\leq u<v\leq n}\{L^k(u,v) + d_{\mathfrak{S}}(u,v)\}. \tag{2.4}$$

The values of $L^i(u, v)$ are initialised by $-\infty$ to prevent the case where there are less than $i - 2$ vertices between u and v.

To implement the method, first calculate $L^2(u, v), L^3(u, v), \ldots, L^k(u, v)$ for all pairs $u < v$ using (2.3), and then compute l^k_{max} using Equation (2.4). Note that Equation (2.3) does not require a recursive implementation. Instead, the results computed in the previous step are stored in such a way that they can be looked up efficiently later on. The results for ordered 2-paths is the split distance matrix itself. Once the results for ordered i-paths are available, they will be used together with the split distance matrix to find the results for ordered $(i + 1)$-paths. The memory space used for ordered i-paths can then be released.

We finish the section by briefly discussing the time and memory consumption of the resulting algorithm. Considering all pairs u and v requires $O(n^2)$ time. For each pair, choosing the element preceding v requires $O(n)$ time. All the above have to be done for the length from 2 to k. So constructing all $L^i(u, v)$ using Equation (2.3) requires $O(kn^3)$ time. Choosing a maximum k-tour from the $O(n^2)$ maximum k-paths using Equation (2.4) requires $O(n^2)$ time. The total running time is thus $O(kn^3)$.

For the memory requirement, from Equation (2.3), it can be seen that for a fixed starting point u, to calculate $L^i(u, v)$, we only need to refer to $L^{i-1}(u, v)$

and the original split distance matrix. Thus, we can compute $L^i(x_1, v)$, infer the longest circular k-tour containing x_1. After that, the memory space can be reused to calculate the longest k-tours containing $x_2, \ldots, x_{n-k+1}$, respectively. For a fixed starting point u, we need to consider $O(n)$ ending points, for each endpoint v, we need $O(k)$ storage for the ordered i-path from u to v. Therefore, the total memory requirement is $O(kn)$.

2.5 Concluding remarks

We have introduced the theoretical foundations underlying the research presented in this thesis, namely phylogenetic trees, split systems, split networks and phylogenetic diversity. Splits are building blocks for all of these and are a powerful tool for studying the evolutionary history of species. In the following chapters, based mostly on splits, we will present some new results to the MPD problem considered in different situations. We will also formally introduce a new kind of split system that can be visualised by a planar split network and make some initial investigation of its applications.

Chapter 3

Hardness results

3.1 Summary

In this chapter we first briefly review the concept of NP-hardness. After that we prove that the general Maximum Phylogenetic Diversity problem introduced in Section 2.4.2 is NP-hard. Finally, we present some hardness results on the optimisation problems with several other measures of diversity.

The following results on problems MPD, WAPD, MED, MO and MSS are adapted from the paper (136). The analysis of problem MAD, and the heuristic algorithm for problem MSS presented in this chapter are additional results.

3.2 Basic complexity concepts

We used the term of "NP-hardness" in Chapter 2 to indicate that it is unlikely that for certain problems, there exist efficient algorithms to solve them. We also used the big-O notion to represent the running time and memory consumption of an algorithm. In this section, we introduce these concepts more formally following the books by Cormen *et. al.* (33) and Garey and Johnson (60).

In computational complexity theory, a *problem* Π is specified by its input and its output. An example of a problem is VertexCover (VC) (33):

VertexCover (VC)
Input: a graph $G = (V, E)$

an integer $k \in \{1, \dots, |V|\}$

Output: the answer to the question "Is there a subset $C \subseteq V$ such that $|C| = k$ and $C \cap e \neq \emptyset$ for all $e \in E$?"

For a $C \subseteq V$, if $C \cap e \neq \emptyset$ for some edge $e \in E$, then we say that C *covers* e; if $C \cap e \neq \emptyset$ for all edges $e \in E$, then we say that C is a *vertex cover* of the graph G. In particular, VC is an example of a so-called *decision problem*, that is, the output is a member of a two-element subset, for example {"yes", "no"} or {"1", "0"}. The *size of the input* usually refers to a quantity that is polynomially related to the amount of memory needed to store the problem's input in a computer. For example, in the VC problem, the size of the input is the number of vertices of the graph.

We say that an algorithm $\mathfrak{A}$ *solves* a problem Π if for every input to that problem, $\mathfrak{A}$ computes a correct output. The complexity of such an algorithm $\mathfrak{A}$ is $O(f(n))$, or for short, algorithm $\mathfrak{A}$ is $O(f(n))$, where $f(n)$ is a function on the size n of the input, if there exists a pair of positive integers C, K such that the number of operations performed by $\mathfrak{A}$ is at most $C * f(n)$ for all inputs of size $n \geq K$. Similarly, the memory consumption of an algorithm can also be measured. We say that algorithm $\mathfrak{A}$ takes $O(g(n))$ memory if there exist a pair of positive integers C', K' such that the amount of memory required by $\mathfrak{A}$ on any input of size n is at most $C' * g(n)$ for all $n \geq K'$.

We now give an overview of several complexity classes that are related to the results in this thesis. We say that problem Π is in the class P, or Π is *polynomially solvable*, if there is an algorithm $\mathfrak{A}$ to Π such that $\mathfrak{A}$ is $O(n^k)$ where n is the size of the input and k is some positive integer.

In the remainder of this section, if not stated otherwise, we only deal with decision problems. A problem Π is in the class NP (Nondeterministic Polynomial) if it is *polynomially verifiable* (33). For example, considering the problem VC, given a k-element subset C of V, it is easy to check in polynomial time if $C \cap e \neq \emptyset$ for all $e \in E$. So VC is in NP.

The class NP-Hard contains problems that are "as hard as" any problem in NP in the sense that if *any* NP-Hard problem is polynomially solvable, then *every* problem in NP can be solved in polynomial time. More formally, a problem Π

is NP-hard if every problem $\Pi' \in$ NP can be *reduced* to Π, that is, there exists a polynomial time algorithm $\mathfrak{A}'$ that computes for every input α of Π' an input $\mathfrak{A}'(\alpha)$ of Π so that the output for Π' with input α is "yes" if and only if the output for Π with input $\mathfrak{A}'(\alpha)$ is "yes". In that case, Π is also said to be *polynomially reducible*, or shorter, *reducible*, from Π', and the algorithm $\mathfrak{A}'$ is called a *reduction* from Π' to Π.

The immediate consequence of a problem Π being reducible from a problem Π' is that if there is a polynomial time algorithm $\mathfrak{A}$ to solve Π, then Π' can be solved by a polynomial time algorithm $\mathfrak{A}^*$. This algorithm $\mathfrak{A}^*$ works by first transforming an input α for Π' to an input for Π via $\mathfrak{A}'$, and then applying $\mathfrak{A}$ on Π with input $\mathfrak{A}'(\alpha)$. As a result, one way to prove that a problem Π is NP-Hard is to choose a problem Π' that has already been known to be NP-Hard, and then design a reduction from Π' to Π. If a problem is in NP and is NP-Hard, then it is called NP-Complete. VC is also an example of an NP-Complete problem (33).

Many problems arising in applications are not decision problems but are *optimisation problems*, i.e. problems where the aim is to optimise (maximise or minimise) a so-called *object function*. For instance, a well known optimisation problem in graph theory is MinimumVertexCover (MVC) (60):

MinimumVertexCover (MVC)
Input: a graph $G = (V, E)$
Output: a subset $C \subseteq V$ of minimum cardinality such that $C \cap e \neq \emptyset$ for all $e \in E$

Here the object function is the size of the subset C. Usually, it is straightforward to associate a decision problem with a given optimisation problem such that if there is a polynomial time algorithm to solve the optimisation problem, then there is also a polynomial time algorithm for the decision problem. For example, clearly any polynomial time algorithm for MVC yields a polynomial time algorithm for VC: Let C be an output for MVC, if $|C| \leq k$ then the output for VC is "yes", otherwise it is "no". As a consequence, we say that an optimisation problem is NP-Hard if the corresponding decision problem is NP-Hard. In particular, MVC is NP-Hard. It is noticeable that even though the two classes

NP and NP-Complete are restricted to decision problems, the class NP-Hard can contain non-decision problems.

3.3 Hardness results

3.3.1 The Maximum Phylogenetic Diversity problem

We now apply the concepts introduced just above to study the complexity for some optimisation problems arising in the study of phylogenetic diversity. Let X be a finite set of n elements. Let $\mathfrak{S}(X) = \{\{A, X \setminus A\} | \emptyset \subset A \subset X\}$ be a split system on X, and let $\omega : \mathfrak{S} \to \mathbb{R}_{>0}$ be a split weight function on $\mathfrak{S}$. Referring to the MPD problem defined in Section 2.4.2, we prove the following theorem.

Theorem 3. *MPD is NP-hard.*

Proof. To show the NP-hardness of the optimisation problem MPD, we consider the corresponding decision problem DMPD which is defined as follows:

DecisionMaximumPhylogeneticDiversity (DMPD)

Input:	a set X with n elements
	a split system $\mathfrak{S}$ on X
	a split weight function $\omega : \mathfrak{S} \to \mathbb{Z}_{>0}$
	an integer $p \in \{1, \ldots, n\}$
	a positive integer B
Output:	the answer to the question "Is there a subset $Y \subseteq X$ such that $\|Y\| = p$ and $PD(Y) \geq B$?"

We will show that DMPD is NP-Hard by a reduction from VC. We describe the algorithm to transform an input of VC to an input of DMPD: Let $X = V$, we associate to G the split system $\mathfrak{S}_G = \{\{e, V \setminus e\} | e \in E\}$ together with a split weight function ω_G that assigns 1 to every split in $\mathfrak{S}_G$. By this construction, for a subset C of V, $|C| \geq 3$, $PD(C)$ equals the number of edges that are covered by C. Since it can be checked easily in polynomial time whether there exists a 2-element subset of V that covers all edges of G, we will assume that $k \geq 3$. Let $p = k, B = |E|$. Then, it is not hard to see that the output for the given input of VC is “yes” if and only if the output for the constructed input for DMPD is “yes”. □

Note that even MPD is NP-hard, a greedy approach similar to that proposed by Faith (47), i.e. start with a two-element subset of X with maximum diversity, then sequentially add $k-2$ elements, such that in every addition the increase in the diversity is maximum, is guaranteed to obtain a k-element subset Y' of X such that $PD(Y')$ is at least $(1-\frac{1}{e})$ times the maximum possible PD of a k-element subset (78).

We now shed some light on the complexity of dealing with data sets arising from multiple regions of a genome by addressing a problem that was recently raised in (106, p.772). In this problem, one wants to compute optimal PD subsets in case one is given several trees on the set X (where e.g. each tree is derived from a different genome region). Formally, the problem is as follows:

WeightedAveragePD (WAPD)

Input:	a set X with n elements
	a collection $\mathfrak{C}$ of compatible split systems on X
	split weight functions $\omega_{\mathfrak{S}} : \mathfrak{S} \to \mathbb{R}_{>0}$, $\mathfrak{S} \in \mathfrak{C}$
	an integer $k \in \{1, \dots, n\}$
Output:	a subset $Y \subseteq X$ with the property that $\lvert Y\rvert = k$ and $\frac{1}{\lvert\mathfrak{C}\rvert}\sum_{\mathfrak{S}\in\mathfrak{C}} PD_{\mathfrak{S}}(Y) = \max\{\frac{1}{\lvert\mathfrak{C}\rvert}\sum_{\mathfrak{S}\in\mathfrak{C}} PD_{\mathfrak{S}}(X') \lvert X' \subseteq X, \lvert X'\rvert = k\}$

It is straight-forward to show that this problem is equivalent to solving MPD on the split system

$$\mathfrak{S}' = \cup_{\mathfrak{S}\in\mathfrak{C}}\mathfrak{S}$$

with split weight function $\omega' : \mathfrak{S}' \to \mathbb{R}_{>0}$ defined by

$$\omega'(S) = \sum_{\mathfrak{S}\in\mathfrak{C}, S\in\mathfrak{S}} \frac{\omega_{\mathfrak{S}}(S)}{\lvert\mathfrak{C}\rvert}$$

for all $S \in \mathfrak{S}'$. Since, conversely, every split system can be written as the union of a collection of compatible split systems, the problems MPD and WAPD are in fact equivalent. So we have:

Theorem 4. *WAPD is NP-hard.*

Note that it can be shown by reduction from MAXIMUM3DIMENSIONALMATCHING (60) that WAPD is NP-hard even when collection $\mathfrak{C}$ contains only three compatible split systems. Recently, Bordewich *et. al.* (19) presented an $O(n^3 \log^2 n)$ algorithm to solve the special case of WAPD problem when the collection $\mathfrak{C}$ contains precisely two compatible split systems.

As mentioned in Chapter 2, an important motivation for studying the problem MPD comes from conservation needs and the natural extension from phylogenetic trees to phylogenetic networks. Some variations of MPD for compatible split systems are also studied in (110) where additional constraints must be satisfied. Maybe not surprisingly, many natural constraints lead to computationally hard optimization problems. Indeed, in (110) it is shown that geographic constraints can lead to NP-hard problems (the OPTIMIZING DIVERSITY WITH REGIONS and the OPTIMIZING DIVERSITY WITH COVERAGE problems); see also (18). However, as reported in (124), integer linear programming can be of use in solving such problems.

It is worth noting here that another variation, the OPTIMIZING DIVERSITY WITH DEPENDENCIES problem considered in (110) is also NP-hard. In this problem, we are given a compatible split system $\mathfrak{S}$ on X and an acyclic digraph $D = (X, A)$. A subset Y of X is *viable* if for every $x \in Y$ vertex x in D has either out-degree 0 or there exists some $y \in Y$ such that (x, y) is an arc in D. The goal is to compute a k-element subset of X that maximizes $PD_{\mathfrak{S}}$ among those k-element subsets of X that are viable. We can form the corresponding decision problem for this by changing the goal to the question "Is there a viable subset of k so that its PD score is greater C?" for a pre-defined given constant C. For this problem the NP-hardness can be shown by a reduction from MINIMUMVERTEXCOVER.

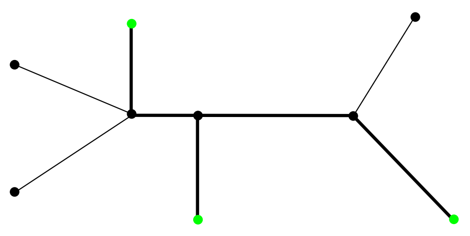
Another variation of the MPD problem is the Noah's Ark Problem, which was proposed by Weitzmann (147). In this problem, a cost for boosting the probability of survival of a species is given and the goal is to maximize the expected PD subject to a given budget. It has been remarked (76) that the Noah's Ark Problem is a generalisation of the well known Knapsack problem (60) and thus is NP-hard.

However polynomial solutions have been developed for the problem considered on phylogenetic trees (108; 117) and circular split systems (108).

To end this section, we would like to point out that, besides PD, several other measures of diversity have been proposed in the literature, see for example (124; 130). These measures also yield computationally interesting optimisation problems. In the next sections we examine the optimisation problems attached to some of the measures and prove that they are NP-hard.

3.3.2 The exclusive molecular phylodiversity

The *exclusive molecular phylodiversity*, denoted as ED, was proposed by Lewis and Lewis (97) in 2005. On a phylogenetic tree on taxa set X, the exclusive molecular phylodiversity of a subset Y of taxa is the total weight of all the edges that support either individual taxa or clades composed exclusively of taxa in some group of interest. In other words, it is the sole contribution of Y on the diversity of the whole tree, or the total weight of those edges that are not in the subtree spanned by $X \setminus Y$. Figure 3.1 illustrates the ED measure in comparison with PD on a tree.


(a) PD

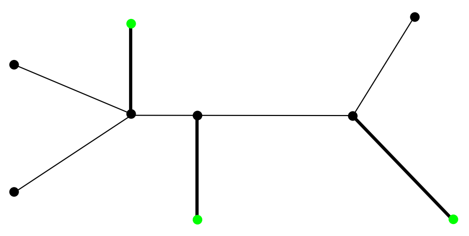

(b) ED

Figure 3.1: Y contains the taxa in green, bold edges in 3.1(a) show $PD(Y)$, those in 3.1(b) show $ED(Y)$.

Generalising the measure for a weighted split system $\mathfrak{S}$, we define

$$ED_{\mathfrak{S}}(Y) = PD_{\mathfrak{S}}(X) - PD_{\mathfrak{S}}(X \setminus Y)$$

for every subset Y of X. If there is no confusion about the split system $\mathfrak{S}$ to be used, we write ED instead of $ED_{\mathfrak{S}}$. The ED optimisation problem (MED) is defined as:

MAXIMUMEXCLUSIVEMOLECULARPHYLODIVERSITY (MED)

Input: a set X with n elements
a split system $\mathfrak{S}$ on X
a split weight function $\omega : \mathfrak{S} \to \mathbb{R}_{>0}$
an integer $k \in \{1, \dots, n\}$

Output: a subset $Y \subseteq X$ with the property that $|Y| = k$ and
$ED(Y) = \max\{ED(W)|W \subseteq X, |W| = k\}$

Note that by definition, maximising $ED_{\mathfrak{S}}(Y)$ is equivalent to minimising $PD_{\mathfrak{S}}(X \setminus Y)$. Moulton *et. al.* (110) created a simple example to show that greedy algorithms will not work for MED. Here we go a step further and show that the MED problem is NP-hard.

Theorem 5. *MED is NP-Hard.*

Proof. A similar argument as given in the proof of Theorem 3 can be used, so we only outline the idea. We consider the corresponding decision problem of the ED optimisation problem where an additional integer B is given and the question is: "Is there a subset $Y \subseteq X$ such that $|Y| = k$ and $ED_{\mathfrak{S}}(Y) \geq B$?". The reduction is from the NP-hard decision problem CLIQUE (60). The input for CLIQUE is a graph $G = (V, E)$, a positive integer $p \in \{1, \dots, |V|\}$ and the goal is to decide whether there exists a subset $C \subseteq V$ such that $|C| = p$ and C forms a *clique* in G, that is, every 2-element subset of C is an edge of G. We encode G by the weighted split system $\mathfrak{S}_G$ we introduced in the proof of Theorem 3. The key observation is that, by construction, for a subset C of V, $|C| \leq |V| - 3$, $ED_{\mathfrak{S}_G}(C) = PD_{\mathfrak{S}_G}(V) - PD_{\mathfrak{S}_G}(V \setminus C) = |E| - PD_{\mathfrak{S}_G}(V \setminus C)$ equals the number of edges in E with both endpoints in C. Hence we set $B = \begin{pmatrix} k \\ 2 \end{pmatrix}$ and $k = p$, then an answer "yes" for the constructed instance of MED yields an answer "yes" for the given instance of CLIQUE and vice versa. □

3.3.3 The M score for diversity

A somewhat different measure of diversity is inspired by a dissimilarity-based method proposed for model strain selection (79, Section 5.3). In that method, given a set of individuals of a single species, to choose one individual as the set's representative, it is suggested to take the one whose maximum distance to the other individuals is minimum. Contextually, that minimum maximum distance could be considered an indicator of the "closeness" of the individuals. In contrast, to quantify the diversity of the set, we can choose the maximum of the minimum pairwise distance between the individuals. That is, for X a set of taxa, we set

$$M_{\mathfrak{S}}(Y) = \min\{PD_{\mathfrak{S}}(\{u,v\}) | \{u,v\} \subseteq Y^2\}$$

for every subset Y of X. In other words, $M_{\mathfrak{S}}(Y)$ is the minimum pairwise distance among elements of Y. If there is no confusion about the split system to be used, we will write $M(Y)$ instead of $M_{\mathfrak{S}}(Y)$. Then the M optimisation problem (MO) is formally defined as follows:

M-Optimisation (MO)

Input: a set X with n elements
a split system $\mathfrak{S}$ on X
a split weight function $\omega : \mathfrak{S} \to \mathbb{R}_{>0}$
an integer $k \in \{1, \ldots, n\}$

Output: a subset $Y \subseteq X$ with the property that $|Y| = k$ and
$M(Y) = \max\{M(W)|W \subseteq X, |W| = k\}$

Again, it was shown by an example in (110) that the greedy approach will not find an optimal solution for the MO problem. And again we will prove that MO is eventually NP-hard.

Theorem 6. *MO is NP-hard.*

Proof. Again, as the proof is similar to the proof of Theorem 3, we only outline the idea. We consider the corresponding decision problem of the M optimisation problem where an additional integer B is given and the output is the answer to the question: "Is there a subset $Y \subseteq X$ such that $|Y| = k$ and $M_{\mathfrak{S}}(Y) \geq B$?". The reduction is from the NP-hard IndependentSet problem in graphs where all vertices have degree three (60). Given such a graph $G = (V, E)$ and a positive integer $p \in \{1, \ldots, |V|\}$, the goal is to answer if there exists a subset $C \subseteq V$ such that $|C| = p$ and C forms an *independent set* in G, that is, no edge in E has both endpoints in C. We encode G by the weighted split system $\mathfrak{S}_G$ we introduced in the proof of Theorem 3. Here the key observation is that, due to the fact that every vertex in G has degree three, for a subset C of V, $|C| \geq 2$, $M_{\mathfrak{S}_G}(C) = 6$ if and only if C is an independent set in G. Now we set $p = k$ and $B = 6$, then an answer "yes" for the constructed instance of MO yields an answer "yes" for the given instance of IndependentSet and vice versa. □

3.3.4 The number of segregating sites

In this subsection we show how PD on split systems can also be used to shed light on intra-species diversity. The *genetic diversity* (*GD*) for a subset of a finite, non-empty set of aligned molecular sequences is commonly used to measure the diversity within intra-species studies, and is defined to be the number of *segregating sites* that the subset induces, that is, the number of sites where not all sequences in the subset display the same character (146). If the sequences come from the same gene of individuals of a species, then each segregating site represents a *polymophism*. The corresponding optimisation problem is as follows:

MaximumSegregatingSites (MSS)

Input: a set X of n (gene) sequences of the same length
an integer $k \in \{1, \dots, n\}$

Output: a subset $Y \subseteq X$ with the property that $|Y| = k$ and
Y induces the maximum number of segregating sites
among all k-element subsets of X.

Theorem 7. *MSS is NP-hard.*

Proof. If we restrict our attention to sequences of binary characters, 0 and 1 say, then for a set X of such sequences having length m, the genetic diversity of a subset Y is given by

$$GD(Y) = \sum_{j=1}^{m} \max_{\sigma, \sigma' \in Y} |\sigma(j) - \sigma'(j)|$$

where we consider every $\sigma \in X$ as a map $\sigma : \{1, \dots, m\} \to \{0, 1\}$ in the obvious way. Now, defining the split $S_j = \{\{\sigma \in X | \sigma(j) = 0\}, \{\sigma \in X | \sigma(j) = 1\}\}$ for all $j \in \{1, \ldots, m\}$, the split system $\mathfrak{S}_X = \{S_j | 1 \leq j \leq m\}$, and the split weight function $\omega_X : \mathfrak{S}_X \rightarrow \mathbb{R}_{>0}$ by $\omega_X(S) = |\{j \in \{1, \ldots, m\} | S_j = S\}|$, it can be checked that $GD(Y) = PD_{\mathfrak{S}_X}(Y)$ for any subset Y of X. In particular, it follows that the same reduction from VertexCover presented for MPD in the proof of Theorem 3 can also be used (graph $\Rightarrow$ split system $\Rightarrow$ alignment) to show that the problem of finding a k-element subset of sequences maximising genetic diversity among all k-element subsets of X is NP-hard. $\square$

In intra-species studies, it may be of interest to find good heuristics for computing subsets of high diversity under the measure GD. We propose a simple greedy heuristic inspired by Steel's algorithm to compute a subset of maximum PD on phylogenetic trees (139) as follows: Start with Y containing two sequences with maximum number of segregating sites among all 2-element subsets of X, then incrementally include a sequence that adds the most segregating sites to Y. Interestingly, this very simple algorithm can sometimes be very efficient: For the data set published in (43), it computes an optimal solution for all values of k (exhaustive search was used to check the optimality).

3.3.5 The *AD* score for diversity

In 1983, Tajima (143) proposed mean and variance of the average number of nucleotide differences as a measure of diversity. The mean number of nucleotide differences of a set X of sequences, $\hat{d}(X)$, is defined by

$$\hat{d}(X) = \frac{2}{\binom{|X|}{2}} \sum_{\substack{x,y \in X \\ x \neq y}} d(x, y),$$

where $d(x, y)$ is the number of nucleotide differences between sequences x and y.

It is straight-forward to extend this measure to an arbitrary distance function on X. Given a set X of n taxa, and a distance function $d : X \times X \rightarrow \mathbb{R}_{\geq 0}$, the average pairwise distance of elements of a subset $Y \subseteq X$ is defined as

$$AD(Y) = \frac{2}{\binom{|Y|}{2}} \sum_{\substack{x,y \in Y \\ x \neq y}} d(x, y).$$

Based on the AD measure, the resulting diversity optimisation problem is formally stated as follows:

MaximumAverageDiversity (MAD)

Input:	a set X with n elements
	a distance function $d : X \times X \to \mathbb{R}_{\geq 0}$
	an integer $k \in \{1, \ldots, n\}$
Output:	a subset $Y \subseteq X$ with the property that $\lvert Y \rvert = k$ and
	$AD(Y) = \max\{AD(W) \lvert W \subseteq X, \lvert W \rvert = k\}$

MAD is known to be NP-hard and it is studied under the name of MaximumDispersion in Operations Research (28; 35; 63; 118; 120). In the MaximumDispersion problem, n locations are given and one would like to choose k out of them to build a network of "dispersed" facilities, i.e. the average distance between selected locations is maximised. The distance used in the original MaximumDispersion problem is the Euclidean distance and it was shown that MAD is even NP-hard for that special type of distance (120). The MaximumDispersion problem belongs to a class of multi-facility location problems like the p-median (38; 92), p-center (30; 40), the incapacitated facility location (UFL) (59; 141), etc. Recently, Tamir (144) presented a survey on algorithmic results for this class indicating that in a number of cases, many of which concerning trees, these problems can be solved efficiently.

3.4 Concluding remarks

In this chapter, we have studied optimisation problems on several measures of diversity that have been employed in different contexts and/or for different purposes. Maybe not surprisingly, all measures we considered lead to NP-hard problems. However, as is usual in complexity theory, depending on the actual structure of input data, there may be properties of the input that can be exploited to design efficient algorithms for those problems. In the next two chapters, we present some new results in this direction. In Chapter 4, trees, i.e. compatible split systems, will be studied with PD and AD measures. In Chapter 5, we will propose an efficient method for so-called affine split systems, then make use of this method to develop a new algorithm for circular split systems.

Chapter 4

Optimising diversity on trees

4.1 Summary

In this chapter, we study problem MPD with PD measure and problem MAD in the context of phylogenetic trees, i.e compatible split systems. The PD optimisation problem on trees has been solved in polynomial time in (116; 139) but here we will present an implementation with linear running time, which is obviously optimal. The method makes up one part of the paper (136). The MAD problem has been shown to be NP-hard in the previous chapter but by making use of the tree structure, we will also develop a polynomial time algorithm for solving it.

4.2 Computing phylogenetic diversity for compatible split systems

The problem of computing sets of maximum PD on phylogenetic trees and some approaches to solve it have been introduced in Section 2.4.1. Here we use the idea presented in (133) for computing tree cores to derive an algorithm with run time $O(n)$. Interestingly, this run time is independent of k and clearly optimal.

To describe our algorithm we assume that the compatible split system $\mathfrak{S}$ is given as an edge-weighted phylogenetic tree $T = (V(T), E(T))$ on a leaf set X, no vertices of degree two and a function that assigns a non-negative weight to each edge of T. This assumption was also made in (106).

The following notation will help simplify the description of the algorithm. Let $L(T) = X$ denote the leaf set of T. For any pair of vertices u and v of T let $\pi(u, v)$ denote the unique path in T between u and v, and $d(u, v)$ the sum of the weights of the edges along path $\pi(u, v)$, also referred to as the *length* $\lambda(\pi(u, v))$ of path $\pi(u, v)$. We denote the vertices on path π by $V(\pi)$. For any non-empty proper subset $U \subset V(T)$ and any $v \in V(T) \setminus U$ we also define $d(U, v) = \min\{d(u, v)|u \in U\}$.

We now recall Faith's greedy algorithm (47) which is illustrated in Figure 4.1: First select two leaves l_1 and l_2 of T such that $d(l_1, l_2)$ is maximized. Let $\pi_1 = \pi(l_1, l_2)$. Next select a leaf $l_3 \in L(T) \setminus \{l_1, l_2\}$ such that $d(V(\pi_1), l_3)$ is maximized. Note there exists a unique vertex u in $V(\pi_1)$ such that $d(V(\pi_1), l_3) = d(u, l_3)$ and the path $\pi_2 = \pi(u, l_3)$ has no edge in common with π_1. Now iterate this procedure. At iteration i, $1 \leq i \leq n - 2$, we have a collection $\beta_i = \{\pi_1, \dots, \pi_i\}$ of pairwise edge-disjoint paths. Define $U_i = \cup_{j=1}^{i} V(\pi_j)$ and the set of leaves $L_i = \{l_1, l_2, \dots, l_{i+1}\}$ selected so far. Select a leaf $l_{i+2} \in L(T) \setminus L_i$ such that $d(U_i, l_{i+2})$ is maximized. As above, there exists a unique vertex u in U_i such that $d(U_i, l_{i+2}) = d(u, l_{i+2})$ and path $\pi_{i+1} = \pi(u, l_{i+2})$ has no edge in common with any of the paths in β_i. This completes the description of the greedy algorithm.

It has been shown (116; 139) that for every $k \in \{2, \dots, n\}$ the set L_{k-1} is an optimal solution for MPD. Note that $\lambda(\pi_1) \geq \lambda(\pi_2) \geq \cdots \geq \lambda(\pi_{n-1})$ and $PD(L_{k-1}) = \sum_{j=1}^{k-1} \lambda(\pi_j)$. Thus, once we have computed the collection $\beta_T := \beta_{n-1}$ of paths, we only need to select for the given k the $k - 1$ longest among them. But this can be done in $O(n)$ time, independently of k (16). It therefore only remains to describe how to compute β_T in $O(n)$ time.

A path π of maximum length in T can be found in $O(n)$ time using a breadth first search procedure, see e.g. (145). We then select an arbitrary endpoint r of π. Vertex r is a leaf of T and we root the tree T at r. This induces a parent-child relation on the vertices of T, and so we refer to the subtree T_u rooted at an arbitrary vertex u of T. We now recursively define for each subtree T_u a collection of paths $\beta(u)$ and demonstrate that in Figure 4.2.

If T_u consists only of vertex u then we put $\beta(u) := \emptyset$. Otherwise T_u has at least one child. For each child v of u fix an arbitrary path π_v of maximum length in $\beta(v)$. We define π'_v as the path π_v extended by edge $\{u, v\}$. If $\beta(v) = \emptyset$ then

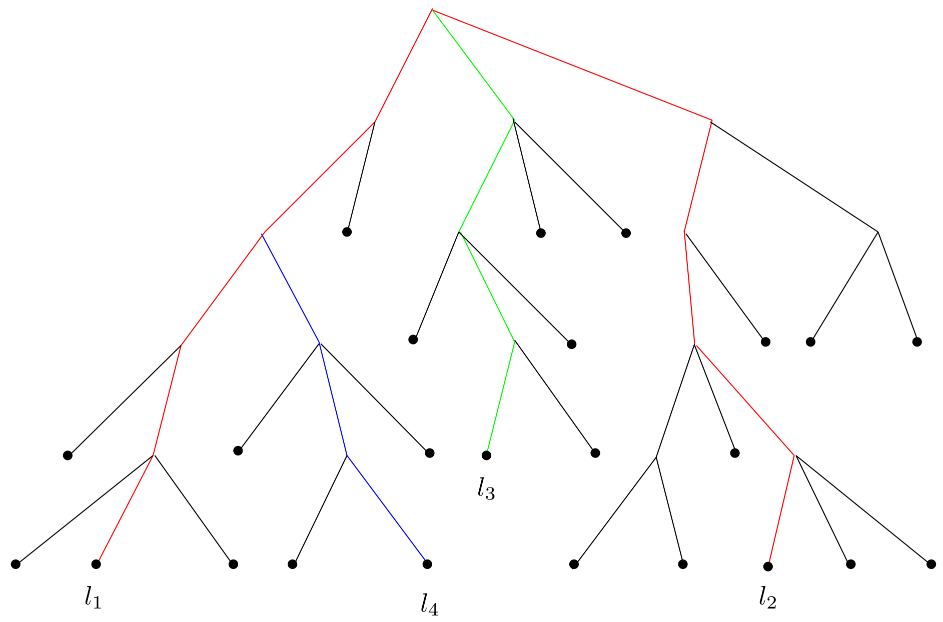


Figure 4.1: The red path π_1 connects the first two leaves to be selected, the green path π_2 adds the third, and the blue path π_3 adds the fourth leaves to the selection.

we define $\pi'_v = \pi(u, v)$. Finally we define the collection of paths $\beta(u)$ as the union of $(\beta(v) \setminus \pi_v) \cup \pi'_v$ over all children v of u.

It can be checked that the resulting collection $\beta(r)$ is a collection of paths that is equivalent to β_T, that is, although $\beta(r)$ and β_T might not contain exactly the same paths, the decreasingly sorted sequence of the lengths of the paths is the same. Using this fact the recursive definition of the collections $\beta(u)$, $u \in V(T)$, can be translated into an easy to implement recursive algorithm. Note that the collections of paths can be organized as linked lists with a pointer to a longest path in each list. With this structure, for every child v_i of a vertex u, we can get $\pi'(v_i)$, and thus $\beta(u)$ in $O(1)$ by linking the $\beta(v_i)$'s altogether, then doing a slight adjustment concerning the longest paths of the $\beta(v_i)$'s to point to the longest path in $\beta(u)$. This results in an algorithm for computing $\beta(r)$ in $O(n)$ time:

Theorem 8. *If the split system $\mathfrak{S}$ is compatible and we are given the corresponding edge weighted phylogenetic tree on the set of taxa X, then MPD can be solved in $O(n)$ time.*

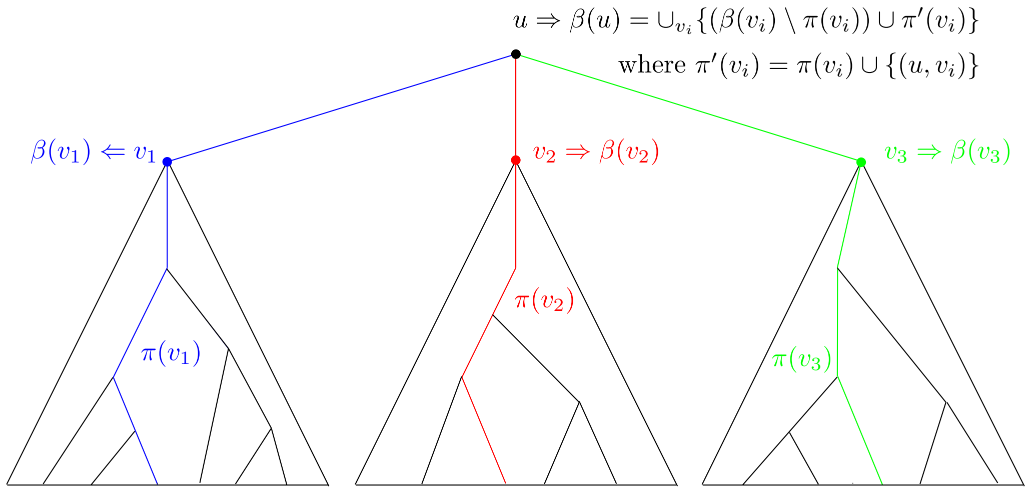

$u \Rightarrow \beta(u) = \cup_{v_i}\{(\beta(v_i) \setminus \pi(v_i)) \cup \pi'(v_i)\}$
where $\pi'(v_i) = \pi(v_i) \cup \{(u, v_i)\}$
$\beta(v_1) \Leftarrow v_1$
$v_2 \Rightarrow \beta(v_2)$
$v_3 \Rightarrow \beta(v_3)$
$\pi(v_1)$
$\pi(v_2)$
$\pi(v_3)$

Figure 4.2: Computing $\beta(u)$ recursively from $\beta(v)$ for all v children of u.

Note that the collection of paths $\beta(T_r)$ needs to be computed only once. After that, it can be queried repeatedly with different values of k.

4.3 An efficient algorithm for MAD on treelike distances

In this section, we present an algorithm based on the dynamic programming technique to solve the *AD* optimisation problem when the distance on X comes from an X-tree. We denote the MAD problem on tree as MAD$_T$.

4.3.1 Preliminaries

Given a weighted X-tree $\mathfrak{T} = (T, \phi)$ where $T = (V, E)$ and a positive integer $k \in \{1, \ldots, n\}$, we have to find a subset $Y \subseteq X$ such that $|Y| = k$ and $AD(Y)$ is maximum among all k-element subset of X, where d is the phyletic distance induced by T between x and y for any $x, y \in X$.

We will transform $\mathfrak{T}$ into a phylogenetic tree by applying two operations. Firstly, we push the labels of interior vertices out to the leaves: For every inner vertex $u \in V$ that is labelled by an element $x \in X$, we remove that label of v and introduce a new edge $\{u, x\}$ with weight 0. Secondly, for every leaf $u \in V$ that is

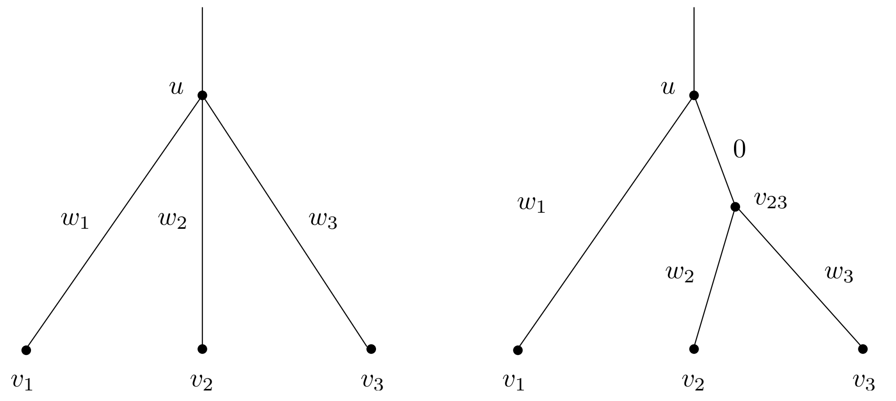


Figure 4.3: Converting a three-child vertex u to two two-child vertices u and v_{23}.

labelled by, say, two elements $x, y \in X$, we remove the labels of v and introduce two new edges $\{u, x\}$ and $\{u, y\}$, both with weight 0. Now a taxon in X can be uniquely identified by the corresponding labelled vertex of T and vice versa.

Next, without loss of generality, we can assume that T is rooted and binary. If T is unrooted, choose an arbitrary edge $e \in E$, insert an additional vertex r to divide e into two new edges e_1, e_2, each of which is assigned a weight of $w(e)/2$ where $w(e)$ is the weight of edge e. If T is not binary, for example if vertex u has three children v_1, v_2 and v_3, see Figure 4.3(a), we can remove two edges $\{u, v_2\}, \{u, v_3\}$, insert a vertex v_{23} and three edges $\{u, v_{23}\}, \{v_{23}, v_2\}$ and $\{v_{23}, v_3\}$, set $w(\{u, v_{23}\}) = 0, w(\{v_{23}, v_2\}) = w(\{u, v_2\}), w(\{v_{23}, v_3\}) = w(\{u, v_3\})$, see Figure 4.3(b). One can check that these modifications do not change the value of $AD(Y)$ for any subset Y of X.

In the following we first describe an algorithm to compute the maximum AD score among all k-element subsets of X. Next we augment this algorithm so that also a k-element subset that yields the maximum AD score is constructed. Our algorithm will be based on a dynamic programming approach. In the next section we define the above mentioned subproblems used in our dynamic programming approach for MAD on a weighted, rooted, binary phylogenetic tree T.

4.3.2 Definition of subproblems

Let Y be an arbitrary subset of X with k elements. For every $w \in V$, we denote by T_w the subtree rooted at w and X_w the set of leaves of T_w. Considering the structure of the tree, it is natural to extend the distance function d on X to the distance between every pair of vertices of V. Let u and v be the children of the root r of T, $Y_u = X_u \cap Y, Y_v = X_v \cap Y$, see Figure 4.4. Note that in the definition of the AD measure given in Section 3.3.5, the factor $\frac{2}{\binom{|Y|}{2}}$ is unchanged among all subsets of size $|Y|$, therefore we will omit that factor in all the following calculations. Let $|Y_u| = l$, we have:

$$\begin{aligned} AD(Y) &= \sum_{x,y\in Y} d(x,y) \\ &= \sum_{x,y\in Y_u} d(x,y) + \sum_{x,y\in Y_v} d(x,y) + \sum_{\substack{x\in Y_u\\ y\in Y_v}} d(x,y) \\ &= AD(Y_u) + (k-l)\sum_{x\in Y_u} d(x,u) + \\ &\quad AD(Y_v) + l\sum_{y\in Y_v} d(y,v) + l(k-l)d(u,v). \end{aligned} \tag{4.1}$$

In Equation (4.1), the quantity $l(k-l)d(u,v)$ does not depend on Y_u and Y_v, and maximising $AD(Y_u) + (k-l)\sum_{x\in Y_u} d(x,u)$ is independent of and similar to maximising $AD(Y_v) + l\sum_{y\in Y_v} d(y,v)$. Therefore if an l-element subset of X_w on the subtree rooted at w that maximises $AD(Y_w) + (k-l)\sum_{x\in Y_w} d(x,w)$ can be computed for every $l \in [0,k]$ and every $w \in V$, then the maximum value of $AD(Y)$ can be computed from Equation (4.1) with the value of l that maximises the sum $\Big(AD(Y_u)+(k-l)\sum_{x\in Y_u} d(x,u)\Big)+\Big(AD(Y_v)+l\sum_{y\in Y_v} d(y,v)\Big)+\Big(l(k-l)d(u,v)\Big)$. This suggests the definition of subproblems in the dynamic programming algorithm as follows.

For every $l \in [0,k]$ and every $u \in V$, we define the score sc of any l-element subset L of X_u as:

$$sc(u,l,L) := \sum_{x,y\in L} d(x,y) + (k-l)\sum_{x\in L} d(x,u). \tag{4.2}$$

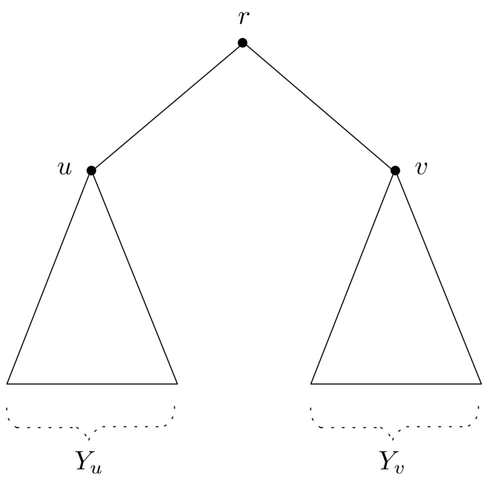


Figure 4.4: Analysing the subproblems of MAD_T.

It is obvious that if $l = k$ then $sc(u, l, L) = AD(L)$. Therefore if we define

$$sc(u, l) := \max \left\{ sc(u, l, L) | L \in \binom{X_u}{l} \right\}, \tag{4.3}$$

then the maximum value of $AD(Y)$ to be found is actually $sc(r, k)$. The value $sc(u, l)$ is the fractional maximum contribution of the subtree rooted at u if l leaves are to be taken from X_u to build up the k-element subset of X with maximum AD. Our subproblems are to compute $sc(u, l)$ for all $u \in V$ and all $l \in [0, k]$.

4.3.3 Structural properties of subproblems

As outlined above, we will compute $sc(u, l)$ for every $u \in V$ and $l \in [0, k]$. The value $sc(u, 0)$ is assigned with 0 meaning that the AD score of an empty set is 0. We now describe how to decompose a subproblem with a given u and a given l into smaller subproblems and when a subproblem is solvable. Examining the subproblem, note that there are three cases to be handled: The first case is that $|X_u| < l$. In this case, there is obviously no l-element subset of X_u, therefore there is no solution to such a subproblem. To indicate this, it is convenient to assign $sc(u, l) = -\infty$ to point out that a subproblem decomposed of this type need not be broken down any further. The second case is that $|X_u| = l$. Then there is a unique solution: X_u is taken and $sc(u,l) = sc(u,l,X_u)$ which can be computed using Equation (4.2). So again, no breaking down into smaller subproblems is necessary here. In the case that $|X_u| > l$, basically we have to consider all l-element subsets L of X_u and use Equations (4.2) and (4.3) to find $sc(u,l)$. However, this would be very time consuming if implemented directly. Here we make use of the optimal substructure of the problem to perform the task more efficiently: Computing $sc(u,l)$ from its children's scores as suggested by the result in the following lemma.

Lemma 1. *For every vertex $u \in V$ such that $|X_u| > 1$, and every $l \in [1, |X_u|-1]$, the following holds:*

$$\begin{aligned} sc(u,l) \quad = \max\Big\{ \quad & sc(v,l_v) + l_v(k-l_v)d(v,u) + \\ & sc(w,l_w) + l_w(k-l_w)d(w,u) \mid \\ & l_v \in [0,l],\ l_w = l - l_v \Big\}, \end{aligned} \tag{4.4}$$

where v, w are the children of u.

Proof. Assume that L is an l-element subset of X_u. Since $|X_u| > 1$, vertex u has two children v and w. Let $L_v = L \cap X_v, l_v = |L_v|, L_w = Y \cap X_w, l_w = |L_w|$. We show how $sc(u,l,L)$ can be built up from the scores concerning v and w:

$$\begin{aligned} sc(u,l,L) \quad &= \quad \sum_{x,y\in L} d(x,y) + (k-l)\sum_{x\in L} d(x,u) \\ &= \quad \sum_{x,y\in L_v} d(x,y) + \sum_{x,y\in L_w} d(x,y) + \sum_{\substack{x\in L_v \\ y\in L_w}} d(x,y) + \\ & \quad\quad (k-l)\left(\sum_{x\in L_v} d(x,v) + l_v d(v,u) + \sum_{y\in L_w} d(y,w) + l_w d(w,u)\right) \\ &= \quad \sum_{x,y\in L_v} d(x,y) + \sum_{x,y\in L_w} d(x,y) + \\ & \quad\quad (k-l)\left(\sum_{x\in L_v} d(x,v) + l_v d(v,u) + \sum_{y\in L_w} d(y,w) + l_w d(w,u)\right) + \\ & \quad\quad l_w \sum_{x\in L_v} d(x,v) + l_v \sum_{y\in L_w} d(y,w) + l_v l_w d(v,w) \end{aligned}$$

$$\begin{aligned}
&= \sum_{x,y \in L_v} d(x,y) + (k-l+l_w) \sum_{x \in L_v} d(x,v) + \\
&\quad \sum_{x,y \in L_w} d(x,y) + (k-l+l_v) \sum_{y \in L_w} d(y,w) + \\
&\quad l_v(k-l)d(v,u) + l_w(k-l)d(w,u) + l_v l_w \Big(d(v,u) + d(w,u) \Big) \\
&= sc(u, l_v, L_v) + l_v(k-l_v)d(v,u) + \\
&\quad sc(w, l_w, L_w) + l_w(k-l_w)d(w,u). \qquad (4.5)
\end{aligned}$$

The last equality follows using the fact that $l_w = l - l_v$. Putting Equation (4.5) into the definition of $sc(u,l)$ in Equation (4.3), we have:

$$\begin{aligned}
sc(u,l) \quad = \max \Big\{ \quad & sc(u, l_v, L_v) + l_v(k-l_v)d(v,u) + \\
& sc(w, l_w, L_w) + l_w(k-l_w)d(w,u) \mid \\
& L_v \cup L_w = L, L \in \begin{pmatrix} X_u \\ l \end{pmatrix} \Big\}. \qquad (4.6)
\end{aligned}$$

Note that each subset $L \in \begin{pmatrix} X_u \\ l \end{pmatrix}$ corresponds to a pair of $\{l_v, l_w\}$ (however each pair of $\{l_v, l_w\}$ may correspond to more than one subsets L); moreover, $sc(u, l_v, L_v) + l_v(k-l_v)d(v,u)$ can be maximised independently on $sc(w, l_w, L_w) + l_w(k-l_w)d(w,u)$. Therefore, we can consider all pairs of $\{l_v, l_w\}$ such that $l_v, l_w \geq 0, l_v + l_w = l$, and for each pair, replace $sc(u, l_v, L_v)$ and $sc(w, l_w, L_w)$ by $sc(v, l_v)$ and $sc(w, l_w)$, respectively. Doing this directly yields Equation (4.4) in Lemma 1.

□

Lemma 1 is the key to a dynamic programming algorithm to solve MAD$_T$: The score at u can be computed efficiently from the scores of its children. We solve the problem in a bottom-up fashion: First compute the scores corresponding to all the leaves, then those of the parents of the leaves and so on. We store the values $sc(u,l)$ in a table where we can then look-up later on.

4.3.4 The algorithm

Based on the structural properties established in the previous section, we present an algorithm to solve the MAD_T problem. The input of the algorithm consists of a weighted, binary, phylogenetic X-tree $\mathfrak{T} = (T, \phi)$ with $T = (V, E)$ rooted at vertex r, and an arbitrary integer $k \in \{1, 2, \ldots, n\}$ where $n = |X|$. The output is the maximum AD score among all k-element subset of X. Basically, the algorithm constructs a 2-dimensional array sc with $|V|$ rows corresponding to the vertices of the tree and $(k+1)$ columns corresponding to the possible values for l (from 0 to k). The pseudo-code for this algorithm can be found in Figure 4.5.

The entries of array sc are computed column by column as follows: For the first column, $sc(u, 0)$ is assigned value 0 for every vertex $u \in V$ meaning that the AD score of an empty subset is 0. Then, for each column $l, l \in \{1, 2, \ldots, k\}$, the value $sc(u, l)$ for every $u \in V$ is determined according to one of three cases: In Line 4, if the subtree rooted at u has less than l leaves, then the attempt to take l leaves will always fail. The corresponding entry in sc is assigned $-\infty$ to indicate that any subproblem directly leading to requiring l leaves at the subtree rooted at u will have no result. Lines 5, 6, 7 deal with the situation where the subtree rooted at u has exactly l leaves in which case all the leaves form the l-element subset. Using Equation (4.4) in Lemma 1, we obtain $sc(u, l)$ in Line 7. The last case is examined in Line 8 when $|X_u| > l$. All possibilities of $\{l_v, l_w\}$ ($l_v = [0, l], l_w = l - l_v$) are considered and Equation (4.4) is used to compute $sc(u, l)$. Line 9 returns the maximum score achieved. As the result, we have the following theorem:

Theorem 9. *Algorithm* $\text{S}\small{\text{OLVE}}\text{\_MAD}_T$ *solves the* $\textit{MAD}_T$ *problem in* $O(nk^2)$ *time and requires* $O(nk)$ *memory.*

Proof. The correctness of the algorithm follows directly from the structural properties of the subproblems established in Section 4.3.3, in particular from Lemma 1. Next we show the bound on the run time. The loops in lines 2, 3 require $O(k)$ and $O(n)$ iterations, respectively, using that $|V| = 2n - 1$ because T is binary (33). Lines 1, 4, 5, 6, 7 all run in constant time. In Line 8, since $l + 1$ possibilities of $\{l_v, l_w\}$ need to be considered, assuming that for a given u and l, the value $sc(u, l)$ can be accessed in constant time, then computing $sc(u, l)$ in this case takes

Solve_MAD$_T$

Input: a weighted, rooted, binary phylogenetic X-tree $\mathfrak{T} = (T, \phi)$ where $T = (V, E)$, $|X| = n$, an integer $k \in \{1, 2, \ldots, n\}$

Output: the maximum *AD* score among all k-element subsets of X

```
1. Let $sc(u, 0) := 0\ \forall u \in V$ // initialise sc
2. for all $l \in \{1, 2, \ldots, k\}$ do
3.     for all $u \in V$ (in the order that u is considered after its children) do
4.         if $|X_u| < l$ then $sc(u, l) := -\infty$ // no l-element subset exists
5.         else if $|X_u| = l$ then // there is only one pair of values $(l_v, l_w)$
6.                 if u is a leaf then $sc(u, l) = 0$
7.                 else $sc(u, l) := sc(v, l_v) + l_v(k - l_v)d(v, u)+$
                                 $sc(w, l_w) + l_w(k - l_w)d(w, u)$
                       // where v and w are the children of u
8.         else // $|X_u| > l$, use sc as a lookup table
              $sc(u, l) = \max\{sc(v, l_v) + l_v(k - l_v)d(v, u)+$
                     $sc(w, l_w) + l_w(k - l_w)d(w, u) | l_v + l_w = l; l_v, l_w \geq 0\}$
                       // where v and w are the children of u
9. return $sc(r, k)$
```

Figure 4.5: The algorithm Solve_MAD$_T$.

$O(k) \times O(1)$ time. Returning the maximum AD value in line 9 takes constant time. So the total run time of the algorithm is $O(nk^2)$.

Note that for Line 3, the vertices must have been sorted in an order such that u is considered after its children. However, since T is binary, then $|V| = 2n-1, |E| = 2n-2$, and V can be sorted using a depth-first search (DFS) procedure (33) which runs in $O(|E|) = O(n)$ time. So even taking this into account, the run time of the algorithm is still $O(nk^2)$.

We conclude the proof by showing the bound on the memory consumed by SOLVE_MAD$_T$. The algorithm stores the table sc which take $O(nk)$ storage. The tree T in the input can be stored in three lists: One containing the vertices themselves, one containing the index of a vertex's parent, and one containing the weight of the edge connecting a vertex to its parent. Each of the list will take $O(n)$ storage. So the total memory requirement is $O(nk)$. □

We illustrate the algorithm by an example where the tree T is given in Figure 4.6 and the number of taxa to be chosen is $k = 3$. The scoring table sc of T is built as indicated in Table 4.1. The first column corresponding to $l = 0$ is filled with 0. The value $sc(u, 1) = 0$ for every leaf u as well. To illustrate how an entry of the table is computed in general, consider $u = v_4, l = 2$. There are three possible values for l_v, namely $0, 1$ and 2. If $l_v = 0$ then $l_w = 2$ and we obtain $sc(v_4, l) = sc(v_7, 0) + 0 * 3 * 14 + sc(v_8, 2) + 2 * 1 * 3 = 40$. Similarly, we obtain the scores 52 and $-\infty$ with $l_v = l_w = 1$ and $l_v = 2, l_w = 0$, separately. So $sc(v_4, 2) = 52$ and l_v, l_w are both chosen to be 1. Finally, the algorithm will return the maximum AD score of 118 .

4.3.5 Finding a subset of taxa having the maximum AD score

Referring back to the original AD optimisation problem, we do not only want to compute the maximum AD score but also a k-element subset of taxa that yields the score. Note that there may be more than one such subset.

Examining algorithm SOLVE_MAD$_T$, if at Line 8, the value of l_v that maximises $sc(u, l)$ is stored in a table l_0, which has the same size and indexing strategy as sc, we can use l_0 to trace back and get a subset of taxa that gives rise to the

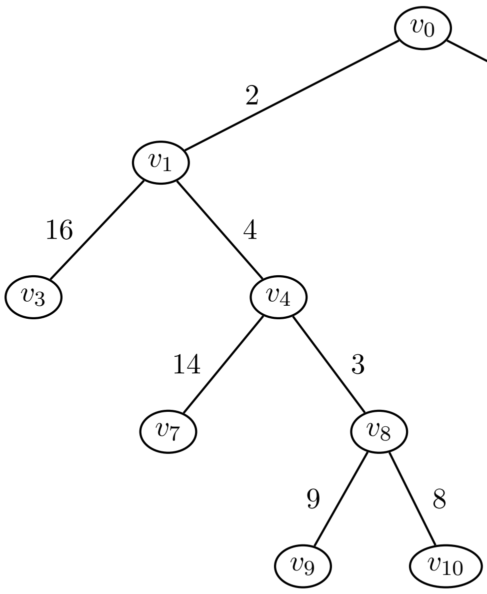


Figure 4.6: The input weighted, binary phylogenetic X-tree T.

	0	1	**2**	3
v_{10}	0	0	$-\infty$	$-\infty$
v_9	0	0	$-\infty$	$-\infty$
v_7	0	0	$-\infty$	$-\infty$
v_6	0	0	$-\infty$	$-\infty$
v_5	0	0	$-\infty$	$-\infty$
v_3	0	0	$-\infty$	$-\infty$
v_8	0	18	34	$-\infty$
v_2	0	32	62	$-\infty$
v_4	0	28	**52**	68
v_1	0	36	68	92
v_0	0	46	86	118

Table 4.1: The scoring table for T with $k = 3$.

maximum AD score following this procedure: Start from entry (r, k), at every cell (u, l), we know that we need to take $l_0(u, l)$ leaves on the subtree rooted at v and $l - l_0(u, l)$ leaves on the subtree rooted at w where v and w are the left and right children of u, we then move to cells $(v, l_0(u, l))$ and $(w, l - l_0(u, l))$. The process on each branch continues until the number of leaves to be selected equals to the number of leaves on the subtree being considered, which means all those leaves will be taken. By this process, a subset of leaves corresponding to a subset of taxa with the maximum AD score can be constructed as the union of the subsets of leaves corresponding to the cases where all leaves of a subtree will be chosen.

We formally define the table l_0 in Equation (4.7) and the algorithm to derive a subset of taxa giving the maximum AD in Figure 4.7.

$$l_0(u, l) = \begin{cases} -2 \text{ if } |X_u| < l, \\ -1 \text{ if } |X_u| = l, \\ \underset{l_v}{argmax}\ sc(u, l) \text{ if } |X_v| > l. \end{cases} \tag{4.7}$$

The entries in table l_0 show how to process with the backtracking: The value $l_0(u, l) = -2$ is just for the completion of the table because the backtracking process never goes to such an entry. If $l_0(u, l) = -1$, then all the leaves of the subtree rooted at u are to be taken. Otherwise, $l_0(u, l) \in [0, l]$ indicates the number of leaves to be taken from the left subtree of u; accordingly, $l - sc(u, l)$ indicates the number of leaves to be taken from the right subtree of u. Table l_0 can be built up along with table sc and this does not change the complexity of algorithm SOLVE_MAD$_T$.

On complexity, algorithm FINDSUBSET_MAD$_T$ uses the stack data structure (33) to implement the procedure described above: It starts at the root of the tree and identifies the number of leaves to be collected from the left and right subtrees. The process continues with every subtree involved. Therefore the run time of the algorithm is linear to the maximum size of the stack, which is the number of subtrees considered. Note that each internal vertex is pushed into the stack at most once. Since T is a binary tree, the number of internal vertices does not exceed the number of leaves (33); therefore, the size of the stack is bounded

FindSubset_MAD$_T$

Input: as in Algorithm Solve_MAD$_T$ and table l_0 as defined above
Output: a k-element subset Y of X such that $AD(Y)$ is maximum among all k-element subsets of X.

```
1.  Let $Y = \emptyset$
2.  Let $Stack = \emptyset$
3.  **push**(Stack, (r,k))
4.  **while** $Stack \neq \emptyset$
5.      **pop**($Stack, (u, l)$)
6.      **if** $l_0(u, l) = -1$ // *the case where* $l = |X_u|$
            **then** $Y = Y \cup X_u$
7.      **else** // *Note that the backtracking never drives to a* $-\infty$ *cell in sc,*
            // *thus never goes to a -2 cell in* $l_0$
8.          **if** $l_0(u, l) > 0$ **then** // *Will collect some leaves on the left subtree*
                **push**($Stack, (v, l_0(u, l))$)
9.          **if** $l_0(u, l) < l$ **then** // *Will collect some leaves on the right subtree*
                **push**($Stack, (w, l - l_0(u, l))$)
10. **return** $Y$
```

Figure 4.7: The algorithm FindSubset_MAD$_T$.

by n. Hence the run time of the algorithm is $O(n)$. The table l_0 requires $O(nk)$ memory.

4.4 Concluding remarks

We have studied the NP-hard problems MPD and MAD in the special case where the input comes from phylogenetic trees, i.e. compatible split systems. The tree structure has been utilised to derive a linear time algorithm for the MPD problem on trees. Not only the running time has been improved compared to currently available methods, but our algorithm is also interesting in that it does not depend on the number of taxa to be selected. For the MAD_T problem, the tree structure has been exploited again, using a dynamic programming approach, to develop a polynomial time algorithm. In the next chapter we will study the MPD problem on split systems having a more complex structure.

Chapter 5

Phylogenetic diversity on circular and affine split systems

5.1 Summary

In Chapter 4, we studied several optimisation problems on split systems related to phylogenetic trees. We now consider the MPD problem with respect to more complicated split systems: Circular and *affine* split systems. We define and investigate affine split systems first, then derive an algorithm for solving the MPD problem for these systems. After that we apply this approach to give a new, more efficient algorithm for the MPD problem on circular split systems. This chapter is extracted and extended from parts of the paper (136).

5.2 Computing phylogenetic diversity for affine split systems

In this section we consider the MPD on so-called affine split systems. As with circular split systems, these split systems have the property that they can be visualized by planar split networks (24) and, as we shall see, solving MPD for such split systems is closely related to the problem of finding optimal convex polygons (44).

We first introduce some notation. Let L be a straight line in the plane with equation $ax + by + c = 0$. We refer to the two open half-planes bounded by L as

$$L^+ = \{(x, y) \in \mathbb{R}^2 | ax + by + c > 0\}$$

and

$$L^- = \{(x, y) \in \mathbb{R}^2 | ax + by + c < 0\}.$$

Let $\mathfrak{L}$ denote the set of all straight lines in the plane. A split system $\mathfrak{S}$ on X is called *affine* if there exist injective mappings $\varphi : X \to \mathbb{R}^2$ and $\theta : \mathfrak{S} \to \mathfrak{L}$ such that for every split $S \in \mathfrak{S}$ we have

$$S = \{\varphi^{-1}(\theta(S)^+ \cap \varphi(X)), \varphi^{-1}(\theta(S)^- \cap \varphi(X))\}.$$

The term "affine split systems" was first defined by Bryant and Dress in their study about linearly independent split systems (22). We will show that if the given split system $\mathfrak{S}$ is affine and we are given corresponding mappings φ and θ then we can solve MPD in $O(kn^3)$ time. To simplify the description we will identify X with the set of points in the plane it is mapped to by φ. We also assume that no three points in X are co-linear.

In order to describe the structure of optimal solutions for MPD we need to introduce some more notation. Let P denote a finite set of points in the plane containing no three collinear points. We let $conv(P)$ denote the convex hull of P, and $H(P)$ denote the set of points in P lying on the boundary of $conv(P)$. Note that the induced distance matrix d of a weighted affine split system can be computed in $O(n^2)$ time using techniques presented in (41).

Lemma 2. *(i) If $|H(X)| = l \leq k$ then, for any $(k - l)$-element subset Z of $X \setminus H(X)$, the set $Y = H(X) \cup Z$ is an optimal solution for MPD.*

(ii) If $|H(X)| > k$ then there exists an optimal solution $Y \subseteq X$ such that the points in Y are in convex position.

(iii) Let $Y \subset X$ be a set of points in convex positions and number the vertices $y_1, y_2, \dots, y_l$ of $conv(Y)$ clockwise around $conv(Y)$. Then $PD_{\mathfrak{S}}(Y) = \frac{1}{2}(d(y_1, y_2) + d(y_2, y_3) + \cdots + d(y_l, y_1))$.

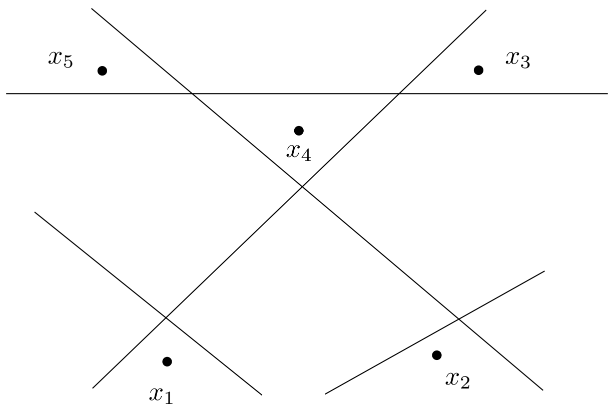


Figure 5.1: When we assign weight 2 to each of the splits in $\{S_1, \dots, S_4\}$ and weight 1 to split S_5, then $PD(\{x_1, x_2, x_4\}) = 8$ and every other 3-element subset has phylogenetic diversity at most 7. But x_4 does not lie on the boundary of the convex hull of the whole point set.

Proof. Property (i) follows from the fact that $PD(Y) = PD(X)$, and, thus, $PD(Y)$ is maximal, for any Y with $H(X) \subseteq Y$.

To show property (ii) let Y be an optimal solution and suppose there exists some $y \in Y$ with the property that y lies in the interior of $conv(Y)$. Since $|H(X)| > k$ there must exist some $y' \in H(X) \setminus Y$. The situation is shown in Figure 5.2(a), $conv(Y)$ is indicated by shading. We define $Y' = (Y \setminus \{y\}) \cup \{y'\}$. Then, since $conv(Y) \subseteq conv(Y')$, we have $PD(Y) \leq PD(Y')$. Illustratively, in Figure 5.2(a), since y lies inside $conv(Y)$, every split that separates y from any of other taxon has its coressponding line intersect with two edges of $conv(Y)$, i.e. that split bi-partites elements of $H(Y)$, or y does not contribute to $PD(Y)$. Clearly, such split contributes to $PD(Y')$. Meanwhile, y' may yield splits that contribute to $PD(Y')$ but not $PD(Y)$, e.g. $\{y'\}|(Y \setminus \{y\})$. Then Y' is an optimal solution that has one more convex point comparing to Y. Iterating this argument yields an optimal solution whose points are all in convex position.

Property (iii) follows from the fact that for a subset $Y \subseteq X$ every straight line L in $\theta(\mathfrak{S})$ that intersects $conv(Y)$ intersects the boundary of $conv(Y)$ at exactly two distinct points. An example is given in Figure 5.2(b), $conv(Y)$ is indicated by shading. □

In view of Lemma 2(i) an optimal solution for MPD can be found easily if

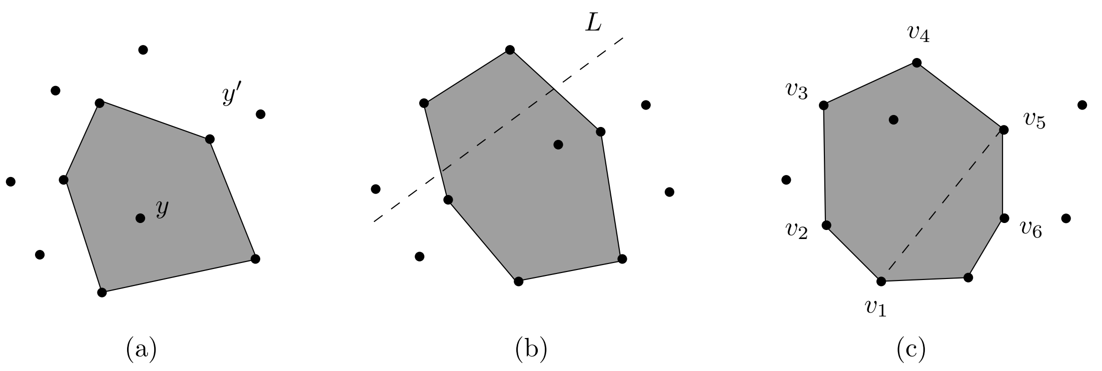


Figure 5.2: Observations on affine split systems.

$|H(X)| \leq k$. Therefore in the following we will assume that $|H(X)| > k$. Then by Lemma 2(ii) and (iii) solving MPD on affine split systems amounts to finding a convex k-gon with vertices in X with maximum perimeter. The problem of finding k-gons with maximum perimeter and vertices in a given set of n points has been considered for the Euclidean distance between the points (20; 44). Here we show that the ideas developed for this problem can be applied to the problem MPD on affine splits.

Before proceeding, we note that there are some differences in the structure of the problems. For example, it is shown in (20) that all the vertices of a convex k-gon with maximum Euclidean perimeter coincide with vertices of the convex hull of the given point set. In contrast, the example shown in Figure 5.1 indicates that this is not always true when we measure the perimeter induced by weighted affine splits. Fortunately, this property is not crucial for devising an efficient algorithm.

Now, let f denote a function that assigns to every convex polygon with vertices in X a non-negative real number. The function f is called *decomposable* (44) if for any convex polygon C with vertices $v_1, v_2, \ldots, v_l$ in clockwise order around C and any $i \in \{3, \ldots, l-1\}$ we have

$$f(C) = \diamond(f(conv(v_1, \ldots, v_i)), f(conv(v_1, v_i, \ldots, v_l)), v_1, v_i),$$

where $\diamond : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \times X \times X \to \mathbb{R}_{\geq 0}$ can be computed in constant time. In addition, f is called *monotone* decomposable if $\diamond$ is a monotone function in its first and second argument. In (44) the following is shown.

Theorem 10 (Eppstein et al. (44))**.** *Let f be a monotone decomposable function on the convex polygons with vertices in X. Then a convex k-gon that maximizes f among all convex k-gons can be computed in $O(kn^3 + \tau(n))$ time, where $\tau(n)$ is the time needed to compute f on all the triangles with vertices in X.*

Corollary 1. *If $\mathfrak{S}$ is an affine split system and we are given the distance matrix d induced by $\mathfrak{S}$ and ω, then MPD can be solved in $O(kn^3)$ time.*

Proof. Using Lemma 2(iii) we first show that $PD_{\mathfrak{S}}$ is monotone decomposable with $\diamond(\xi_1, \xi_2, p, q) = \xi_1 + \xi_2 - d(p, q)$. It is obvious that $\diamond$ is monotone in its first two arguments. Let Y be a set of points in convex positions and number the vertices $y_1, y_2, \dots, y_l$ of $conv(Y)$ clockwise around $conv(Y)$, for any $i \in \{3, \dots, l-1\}$, we have:

$$\begin{aligned} PD_{\mathfrak{S}}(\{y_1, y_2, \dots, y_i\}) + PD_{\mathfrak{S}}(\{y_1, y_i, \dots, y_l\}) - d(y_1, y_i) = \\ \frac{1}{2}(d(y_1, y_2) + d(y_2, y_3) + \dots + d(y_i, y_1)) + \\ \frac{1}{2}(d(y_1, y_i) + d(y_i, y_{i+1}) + \dots + d(y_l, y_1)) - d(y_1, y_i) = \\ \frac{1}{2}(d(y_1, y_2) + d(y_2, y_3) + \dots + d(y_l, y_1)) = PD_{\mathfrak{S}}(Y), \end{aligned}$$

or $\diamond$ is decomposable. An example is given in Figure 5.2(c) where $Y = \{v_1, v_2, \dots, v_7\}$ and $i = 5$. For any of the $O(n^3)$ triangles Δ with vertices in X we can compute $PD_{\mathfrak{S}}(\Delta)$ in $O(1)$ time. Hence, $\tau(n) \in O(n^3)$. Thus, by Theorem 10, MPD can be solved in $O(kn^3)$ time. $\square$

Note that in some applications one might wish to *augment* an affine split system $\mathfrak{S}$ on X by adding all the trivial splits, that is, the splits of the form $\{\{x\}, X \setminus \{x\}\}$, $x \in X$, and giving them positive weights. Note that not every augmented affine split system has the property that it can be visualized by a planar split network. However, it is still possible to solve MPD in polynomial time on augmented affine split systems by adapting the above algorithm that solves MPD for affine split systems, although the bound on the run time increases to $O(k^2n^4)$. The key observation is the following. Let x_1, x_2 and x_3 be three distinct elements in X and $l \geq 3$ some integer. Let Δ denote a triangle in the plane with vertices x_1, x_2 and x_3 with $|X \cap \Delta| \geq l$. Then we can compute in $O(n)$ time an l-element subset Y of X with the property that $PD(Y)$ is maximum among all l-element subsets Y' of X with $\{x_1, x_2, x_3\} \subseteq Y' \subseteq \Delta$. To see this, note that for any $Y' \subseteq X$ with $\{x_1, x_2, x_3\} \subseteq Y' \subseteq \Delta$ the contribution to $PD(Y')$ of an element $y \in Y'$ that lies in the interior of Δ equals the weight of the trivial split $\{\{y\}, X \setminus \{y\}\}$. Therefore, we only need to select $l - 3$ elements of X in the interior of Δ such that the sum of the weights of the corresponding trivial splits is maximized.

5.3 Computing phylogenetic diversity for circular split systems

In this section we use the analysis of affine split systems presented in the previous section for solving MPD on circular split systems. As we shall see, every circular split system corresponds to an affine split system where the points in X are in convex position. Hence, by Corollary 1, MPD on circular split systems can be solved in $O(kn^3)$, the same run time as the algorithm by Minh *et. al.* (107) which is reviewed in Section 2.4.2. In this section we show how geometric considerations can be used to improve this run time to $O(kn + n \log n)$.

In the following we will consider X as a set of points in the plane in convex position and we arrange them in the order $x_1, x_2, \ldots, x_n$ clockwise around $conv(X)$. It is known that the distance matrix d induced by a weighted circular split system satisfies the following four-point condition (31; 88):

$$\begin{aligned} &d(x_{i_1}, x_{i_3}) + d(x_{i_2}, x_{i_4}) \geq d(x_{i_1}, x_{i_2}) + d(x_{i_3}, x_{i_4}) \quad \text{and} \\ &d(x_{i_1}, x_{i_3}) + d(x_{i_2}, x_{i_4}) \geq d(x_{i_1}, x_{i_4}) + d(x_{i_2}, x_{i_3}) \end{aligned} \tag{5.1}$$

for all $i_1, i_2, i_3, i_4 \in \{1, \ldots, n\}$, $i_1 < i_2 < i_3 < i_4$. Clearly, (5.1) holds for the Euclidean distance between points in X, and a careful check of the results in (20) reveals that they rely essentially only on this property. As a consequence the algorithm for finding a convex k-gon with maximum Euclidean perimeter presented in (20), along with the modifications suggested in (5), can also be used to solve MPD on circular split systems. However, the correctness proof given in (20) does not immediately carry over to MPD, since it makes use of the fact that for the Euclidean distance (5.1) holds with both inequalities being strict. For the induced distance matrix d this cannot be guaranteed.

Before we describe the algorithm we require some more notation. In the following, if not stated otherwise, the vertices of every polygon are a subset of X and ordered clockwise around $conv(X)$. For any $u, v \in X$, $u \neq v$, the *interval* $I = [u, v]$ is the set of elements of X we meet when we walk from u to v clockwise along the boundary of $conv(X)$. The vertices u and v are called the *endpoints* of interval I. If $u = v$ then we define $[u, v] = \{u\}$. A convex l-gon C is *rooted at* $r \in X$ if r is a vertex of C. A convex l-gon with maximum perimeter among all l-gons (rooted at r) will be called an *optimal* l-gon (rooted at r).

The algorithm presented in (20) consists of two phases. In the first phase an optimal k-gon rooted at x_1 is computed. In the second phase the result from the first phase is used to compute an optimal k-gon. We now describe our adaptation of the first phase. It is based on the following lemma.

Lemma 3. *Let C be an optimal l-gon rooted at r with vertices $r = v_1, \ldots, v_l$, $l < n$. Then there exists an optimal $(l+1)$-gon C' rooted at r with vertices $r = u_1, \ldots, u_{l+1}$ such that $u_i \in [v_{i-1}, v_i]$ for all $i \in \{2, \ldots, l+1\}$, $v_{l+1} = v_1$.*

Proof. The proof uses the idea of the so called *crossing transform* introduced in (20). We will assume that $l \geq 3$. The case $l = 2$ can be dealt with similarly.

As a first step consider any optimal l-gon C rooted at some vertex r. We want to show that there exists an optimal $(l+1)$-gon C' rooted at r such that every interval induced by C contains at least one vertex of C'. To this end consider an optimal $(l+1)$-gon $\tilde{C}$ rooted at r and suppose that there exists an interval induced by C that does not contain a vertex of $\tilde{C}$. We number the vertices $v_1, \ldots, v_l$ of C in such a way that interval $[v_l, v_1]$ does not contain an element of $\tilde{C}$ but interval $[v_1, v_2]$ does. We number the vertices $u_1, \ldots, u_{l+1}$ of $\tilde{C}$ such that u_1 is the first vertex of $\tilde{C}$ after v_1. An example is given in Figure 5.3(a) where $l = 5$. Define μ_i as the number of vertices of $\tilde{C}$ in $[v_1, v_i]$. Note that $\mu_1 = 0$ and $\mu_l = l + 1$. Let j be minimal with the property that $j \leq \mu_j$. Note also that such a j must exist and that $u_{j-1} \in [v_{j-1}, v_j] \setminus \{v_{j-1}, v_j\}$ and $u_j \in [v_{j-1}, v_j] \setminus \{v_{j-1}\}$. In the example in Figure 5.3(a) we have $j = 4$.

We can now apply the crossing transform, that is, we construct a new l-gon C_1 rooted at r with vertices $u_1, \ldots, u_{j-1}, v_j, \ldots, v_l$ and a new $(l+1)$-gon C_2 rooted at

r with vertices $v_1, \dots, v_{j-1}, u_j, \dots, u_{l+1}$. By the four-point condition (5.1) the sum of the perimeters of C_1 and C_2 is not smaller than the sum of the perimeters of C and $\tilde{C}$. Thus, since both C and $\tilde{C}$ are optimal l- and $(l+1)$-gons, respectively, rooted at r, the perimeter of $\tilde{C}$ and C_2 is the same. Hence, C_2 is an optimal $(l+1)$-gon rooted at r and it can be checked that the number of intervals induced by C that do not contain a vertex of C_2 is strictly less than the number of these intervals for $\tilde{C}$. Thus, by a repeated application of the crossing transform we can construct an optimal $(l+1)$-gon C' rooted at r such that every interval induced by C contains at least one vertex of C'. This finishes the first step of the proof.

Note that with a completely analogous argument we can show that there even exists an optimal $(l+1)$-gon C' rooted at r such that every interval I induced by C contains a vertex of C' that is not an endpoint of I or both endpoints of I are vertices of C'. We will say that such a C' *interleaves nicely* with C. An example of the situation we want to avoid is given in Figure 5.3(b): The interval $[v_5, v_1]$ contains only the vertex $u_6 = v_l$. In contrast, the 5-gon and the 6-gon in Figure 5.3(c) interleave nicely.

In the second step of our proof we show that, given an optimal l-gon C rooted at r with vertices $r = v_1, \dots, v_l$, if an optimal $(l+1)$-gon C' rooted at r interleaves nicely with C then vertex u_i must lie in $[v_{i-1}, v_i]$. To this end, first note that the number of vertices of C' in $[v_1, v_i]$ is at least i. This can shown by induction on i. Similarly, the number of vertices of C' in $[v_i, v_1] \setminus \{v_1\}$ is at least $l - i + 1$. Now suppose for some i vertex u_i does not lie in $[v_{i-1}, v_i]$. Since the number of vertices of C' in $[v_1, v_i]$ is at least i this can only happen if u_i is in $[v_1, v_{i-1}] \setminus \{v_{i-1}\}$ and, thus, at least i vertices of C' lie in $[v_1, v_{i-1}] \setminus \{v_{i-1}\}$. Since the remaining $(l+1)-i$ vertices of C' must all lie in $[v_i, v_1] \setminus \{v_1\}$, this implies that C' does not interleave nicely with C, a contradiction. Hence, vertex u_i must lie in $[v_{i-1}, v_i]$. □

Lemma 3 suggests computing an optimal k-gon rooted at $x_1 = x_{i_1}$ incrementally. First we find an element $x_{i_2} \in X \setminus \{x_{i_1}\}$ such that $d(x_{i_1}, x_{i_2})$ is maximal. Then we repeat adding a new vertex by calling the procedure AddVertex in Figure 5.5 until an optimal k-gon rooted at x_1 is found. Starting with an optimal l-gon rooted at x_1, AddVertex computes an optimal $(l+1)$-gon rooted at x_1. The process is illustrated in Figure 5.4. To do this it employs Lemma 3 to ensure that when searching for the vertices of an optimal $(l+1)$-gon rooted at x_1 it can

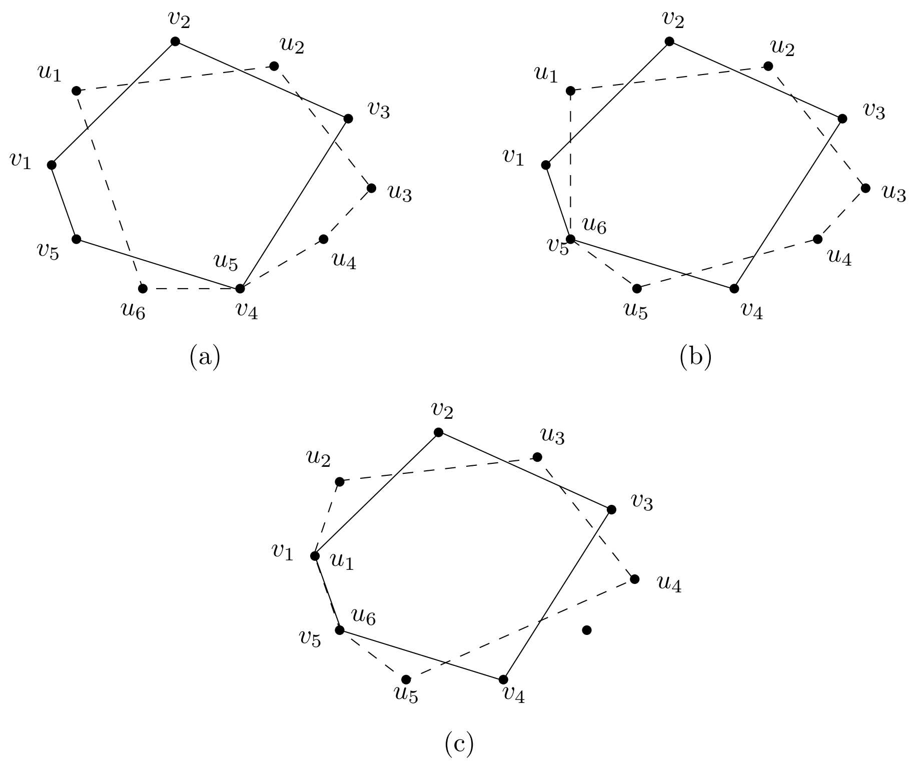
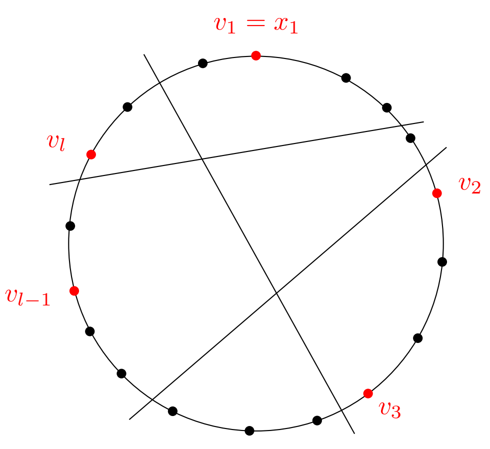


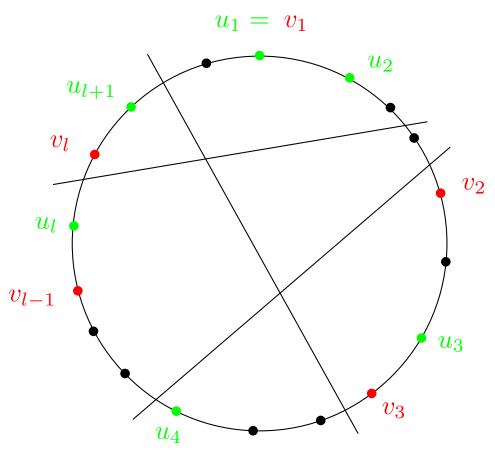
Figure 5.3: Examples used in the proof of Lemma 3.

restrict to the intervals induced by the given optimal l-gon rooted at x_1. To perform this search efficiently the dynamic programming technique is employed: In the loop in Lines 1-2 of the procedure AddVertex the initial tables are built. In general, table $A[j, s]$ is used to store the length of the longest polygonal path from vertex x_1 to vertex x_j under the constraint that this path has $s - 1$ additional vertices besides x_1 and x_j, one in each of the intervals $[x_{i_1}, x_{i_2}], \ldots, [x_{i_{s-1}}, x_{i_s}]$. Table $B[j, s]$ is used to store the index of the vertex, say u, that comes immediately before x_j in the longest path. The nested loops in Lines 3-6 fill in these tables using entries which have already been computed. This can be done since the sub-path from x_1 to u of the longest path from x_1 to x_j must have maximum length, too.

Before we continue to outline the second phase of the algorithm we briefly sketch the analysis of the run time of the first phase. We call procedure AddVertex $(k - 2)$ times. A straightforward implementation of procedure Ad-



(a) An optimal set of l elements $C = \{v_1, v_2, \ldots, v_l\}$. The straight lines illustrate the splits.



(b) An optimal set of $l+1$ elements $C' = \{u_1, u_2, \ldots, u_{l+1}\}$ constructed from C where $u_1 \equiv v_1$, u_i lies between v_{i-1} and v_i for all $2 \leq i \leq (i+1)$, $v_{l+1} \equiv v_1$.

Figure 5.4: Building an optimal set of $(l+1)$ elements from an optimal set of l elements, both sets contain x_1.

dVertex would yield a run time of $O(n^2)$ per call. However, it is possible to apply the techniques presented in (5) to speed up the computations done in Lines 4-6 (Figure 5.5). To this end we define a matrix M_{jt}, $i_s < j \leq i_{s+1}$, $i_{s-1} < t \leq i_s$, by $M_{jt} = d(x_j, x_t) + \lambda(x_t, x_1)$ where $\lambda(x_t, x_1)$ is the length of the longest polygonal path with vertices $x_t = u_s, u_{s-1}, \ldots, u_1 = x_1$, $u_a \in [x_{i_{a-1}}, x_{i_a}]$, $2 \leq a \leq s-1$. We want to find for each row of matrix M the first entry that is larger than or equal to every other entry in the same row. In (5) it is shown that this can be done in $O(\sharp\text{rows of } M + \sharp\text{columns of } M)$ time if M is *totally monotone*, that is, $M_{j_1t_1} < M_{j_1t_2}$ implies $M_{j_2t_1} < M_{j_2t_2}$ for every $j_1 < j_2$ and $t_1 < t_2$. But this follows from the four-point condition (5.1). Note that we do not need to construct matrix M explicitly as each of its entries can be computed in constant time. Thus, procedure AddVertex can be implemented to run in $O(n)$ time. Hence, the first phase of the algorithm can be completed in $O(kn)$ time.

The second phase of the algorithm relies on the result stated in the following lemma which can be proved in a similar way we have shown Lemma 3.

Lemma 4. *Let C be an optimal l-gon rooted at r with vertices $r = v_1, \ldots, v_l$.*

AddVertex($\{x_{i_1}, x_{i_2}, \dots, x_{i_l}\}$)

Input: optimal l-gon $\{x_{i_1}, x_{i_2}, \dots, x_{i_l}\}$ rooted at x_{i_1}, $1 = i_1 < \cdots < i_l \leq i_{l+1} = n$

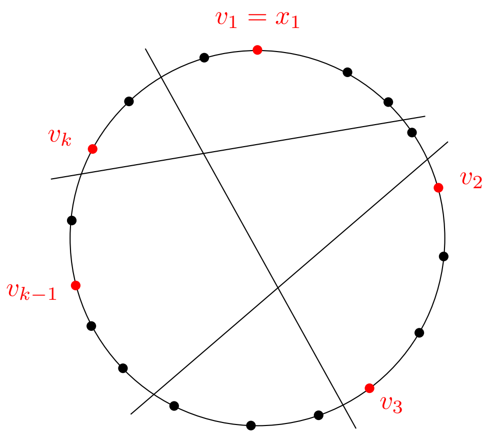
Output: optimal $(l+1)$-gon $\{x_{a_1}, x_{a_2}, \dots, x_{a_{l+1}}\}$ rooted at x_{a_1}, $1 = a_1 < \cdots < a_{l+1} \leq n$

```
1.  for $j := 2$ to $i_2$ do
2.      $A[j,1] := d(x_j, x_{i_1})$, $B[j,1] := i_1$
3.  for $s := 2$ to $l$ do
4.      for $j = i_s$ to $i_{s+1}$ do
5.          $A[j,s] := \max_{\substack{\max\{2,i_{s-1}\} \leq t \\ \min\{i_s, j-1\} \geq t}} d(x_j, x_t) + A[t, s-1]$
6.          $B[j,s] := \underset{\substack{\max\{2,i_{s-1}\} \leq t \\ \min\{i_s, j-1\} \geq t}}{argmax} d(x_j, x_t) + A[t, s-1]$
7.  $a_{l+1} := \underset{i_l \leq t \leq n}{argmax} \; d(x_{i_1}, x_t) + A[t, l]$
8.  for $s := l$ downto $1$ do
9.      $a_s = B[a_{s+1}, s]$
10. return $\{x_{a_1}, x_{a_2}, \dots, x_{a_{l+1}}\}$
```

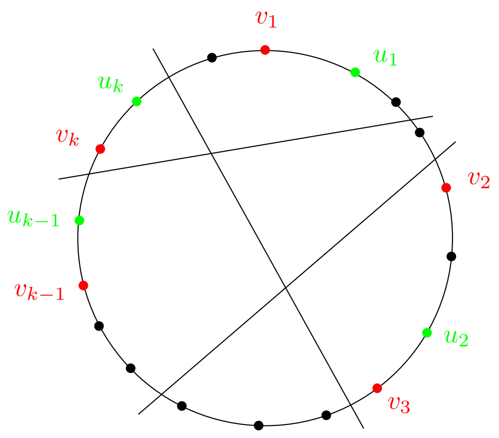
Figure 5.5: The procedure AddVertex. In lines 6 and 7 ties are broken arbitrarily.

Then there exists an optimal l-gon C' with vertices $u_1, \ldots, u_l$ such that $u_i \in [v_i, v_{i+1}]$ for all $i \in \{1, \ldots, l\}$, $v_{l+1} = v_1$.

From the first phase of the algorithm we are given an optimal k-gon C rooted at x_1 with vertices $\{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}$, $1 = i_1 < \cdots < i_k \leq n$. The vertices of C induce k intervals, namely $I_j = [x_{i_j}, x_{i_{j+1}}]$, $1 \leq j \leq k$, $i_{k+1} = i_1$. Figure 5.6 shows the idea employed in the second phase. By Lemma 4 there exists an optimal k-gon where the j^{th} vertex is located in interval I_j. Thus, if we fix some vertex $v \in I_1$, with a procedure that is analogous to the procedure AddVertex we can find a k-gon C_v with maximum perimeter among all the k-gons that have its jth vertex in interval I_j and the first vertex at v. Hence, by checking every vertex $v \in I_1$ we can find an optimal k-gon in $O(n^2)$ time.



(a) An optimal set of k elements $C = \{v_1, v_2, \ldots, v_k\}$. The straight lines illustrate the splits.



(b) A globally optimal set of k elements $C' = \{u_1, u_2, \ldots, u_k\}$ constructed from C where u_i lies between v_i and v_{i+1} for all $1 \leq i \leq k$, $v_{k+1} \equiv v_1$.

Figure 5.6: Building a globally optimal set of k taxa from an optimal set of k taxa containing taxon x_1.

However, in (20) a more efficient way is presented based on the result stated in the following lemma, which can again be shown using a similar argument to the one that we used to prove Lemma 3.

Lemma 5. *Let $I_1, \ldots, I_l$ be a set of intervals with the property that any two intervals have only elements in common that are endpoints of both intervals and* *there exists an optimal l-gon with vertices $u_1, \ldots, u_l$ such that $u_i \in I_i$ for all $i \in \{1, \ldots, l\}$. Let v be an arbitrary vertex in I_1 and C be an l-gon with vertices $v = v_1, \ldots, v_l$ that has maximum perimeter among all l-gons rooted at v and with $v_i \in I_i$ for all $i \in \{1, \ldots, l\}$. Vertex v_i divides interval I_i into two subintervals L_i and R_i with common endpoint v_i where the elements in R_i come after v_i when we walk from v_i clockwise along the boundary of $conv(X)$. Then there exists an optimal l-gon with vertices $u_1, \ldots, u_l$ and either $u_i \in L_i$ for all $i \in \{1, \ldots, l\}$ or $u_i \in R_i$ for all $i \in \{1, \ldots, l\}$.*

Lemma 5 suggests choosing first a vertex v in the middle of interval I_1 and computing the corresponding k-gon C_v. The jth vertex of C_v divides interval I_j into two subintervals. We then iterate this procedure, choosing vertices v_1 and v_2 in each of the subintervals of I_1, computing k-gons C_{v_1} and C_{v_2} and so on. Although in the end we still check every vertex $v \in I_1$, this yields a run time of $O(n \log n)$ for the second phase of the algorithm. In conclusion we have shown the following:

Theorem 11. *If the split system $\mathfrak{S}$ is circular and we are given the induced distance matrix d the problem MPD can be solved in $O(kn + n \log n)$ time.*

5.4 Concluding remarks

We have shown again that, even though computing sets of maximum phylogenetic diversity for weighted split systems is in general NP-hard, if the given split system has a special structure, efficient algorithms do exist. In this case by extending work by previous authors, we have developed a polynomial time algorithm for affine split systems and a new polynomial algorithm for circular split systems. The latter is more efficient than other available algorithms for circular split systems (106; 107).

Both circular and affine split systems are examples of something called Flat Split Systems - the most general class of split systems who can be visualised by planar split networks. We will define and study this new kind of split system in the following chapters. In particular, we will investigate the problem of reconstructing a FSS to represent a distance matrix so as to provide a generalisation of the NeighborNet algorithm.

Chapter 6

Flat split systems

6.1 Summary

In the previous chapters, we have studied the problem of computing PD for several types of split systems, in particular for compatible, circular and affine split systems. A common characteristic of these split systems is that they can be represented by split networks in the plane without crossing edges. In this chapter, we introduce the most general class of split systems, called *flat split systems (FSSs)*, whose corresponding split networks can be drawn in such a way.

Note that the research direction is somewhat changed here. We have been working on computational problems on a variety of split systems but the practical meaning of the algorithms also depends on how such split systems were reconstructed. The overall aim of this chapter and the next one is to build up the methodology around FSSs that can hopefully be used to represent the evolutionary relationships in cases that are problematic for NeighborNet.

We first introduce some objects related to FSSs, namely sequences of permutations, sequences of swapping points, wiring diagrams and p-sequences. Then we define FSSs, briefly explain how an FSS can be visualised by a planar split network, and show that the class of affine split systems are a subclass of the class of FSSs. Finally we define several basic operations to manipulate FSSs. These operations will be building blocks for searching through the FSSs, as described in the next chapter.

6.2 Preliminaries

In this section we introduce some concepts that will be later used to define FSSs.

6.2.1 Sequences of permutations

Definition 1 (Allowable sequence of permutations)**.** *Given a set $X = \{1, 2, \dots, n\}$, an* allowable sequence of permutations *on X is an ordered list of permutations $\Pi = (\pi^0, \pi^1, \pi^2, \dots, \pi^k)$ of X that has the following properties:*

Property 1 (Reversal)**.** *The* initial permutation *is the identity permutation and the last permutation is the reverse permutation of X, i.e. $\pi^0 = (1, 2, \dots, n), \pi^k = (n, n-1, \dots, 1)$.*

Property 2 (Consecutiveness)**.** *For any $i \in \{1, \dots, k\}$, permutation π^i is obtained from its predecessor, π^{i-1}, by the reversal of the order of some consecutive elements of π^{i-1}.*

Property 3 (Uniqueness)**.** *Any two elements $x, y \in X$ are involved in precisely one reversal.*

The term *allowable sequence of permutations*, or *allowable sequence* for short, and its derivatives, were introduced by Goodman and Pollack (69) and has been an effective tool in discrete and computational geometry (15; 54; 68).

For Property 1, in general, the initial permutation π^0 need not necessarily be the identity permutation but can be any permutation of X provided that π^k is still the reverse of π^0.

The consecutiveness property specifically states that if

$$\pi^{i-1} = (\pi_1, \pi_2, \dots, \pi_{p_i-1}, \underbrace{\pi_{p_i}, \pi_{p_i+1}, \dots, \pi_{p_i+t_i-1}}, \pi_{p_i+t_i}, \dots, \pi_k),$$

then

$$\pi^{i} = (\pi_1, \pi_2, \dots, \pi_{p_i-1}, \overbrace{\pi_{p_i+t_i-1}, \dots, \pi_{p_i+1}, \pi_{p_i}}, \pi_{p_i+t_i}, \dots, \pi_k)$$

for some $p_i \in \{1, \dots, k-1\}, t_i \in \{2, \dots, k - p_i + 1\}$.

The uniqueness property means that, for every $i \in \{1, \dots, k\}$, if Δ_i is the set of elements involved in the reversal to get π^{i+1} from π^i, i.e. $\Delta_i = \{\pi_{p_i}, \pi_{p_i+1}, \dots, \pi_{p_i+t_i-1}\}$

in the definition of the consecutiveness property, then for any $x, y \in X, x \neq y$, there is one and only one $i \in \{1, \dots, k\}$ such that $x, y \in \Delta_i$.

To illustrate these definitions, we give an example of an allowable sequence together with elements involved in each reversal.

Example 1. *An allowable sequence of permutations:*

$$(1,2,3,4,5) \xrightarrow{3,4} (1,2,4,3,5) \xrightarrow{1,2,4} (4,2,1,3,5) \xrightarrow{1,3,5} (4,2,5,3,1) \xrightarrow{2,5} \\ (4,5,2,3,1) \xrightarrow{4,5} (5,4,2,3,1) \xrightarrow{2,3} (5,4,3,2,1).$$

Here the numbers above the arrows indicate those that are involved in the reversal, i.e. $\Delta_1 = \{3,4\}, \Delta_2 = \{1,2,4\}, \Delta_3 = \{1,3,5\}, \Delta_4 = \{2,5\}, \Delta_5 = \{4,5\}, \Delta_6 = \{2,3\}$.

When each reversal in the allowable sequence involves only two elements, we call it a *simple allowable sequence (of permutations).* More formally,

Definition 2 (Simple allowable sequences)**.** *Given a set $X = \{1, 2, \dots, n\}$, an allowable sequence $\Pi = (\pi^0, \pi^1, \pi^2, \dots, \pi^k)$ on X is called a* simple allowable sequence (on X) *if each reversal from π^{i-1} to π^i involves only two elements, i.e. $|\Delta_i| = 2$ for every $i \in \{1, \dots, k\}$, and the reversal is just the swap of two consecutive elements (π_{p_i}, π_{p_i+1}).*

We also call such a sequence a simple allowable sequence of size n. Given a simple allowable sequence of size n, each value p_i is called a *swapping point.* The list $P = (p_1, p_2, \dots, p_k)$ is an example of a sequence of swapping points of size n. We will formally define a sequence of swapping points after introducing the construction of a list of permutations as follows: Let π^0 be an arbitrary permutation of the set $\{1, \dots, n\}$, $P = (p_1, p_2, \dots, p_k)$ be an ordered list of numbers where $p_i \in \{1, 2, \dots, n-1\}$ for all $i \in \{1, \dots, k\}$. We construct a list of permutations $\Pi = (\pi^0, \pi^1, \dots, \pi^k)$ where permutation $\pi^i, i \in \{1, \dots, k\}$, is derived from π^{i-1} by putting:

$$\begin{aligned} \pi^i_{p_i} &= \pi^{i-1}_{p_i+1}, \\ \pi^i_{p_i+1} &= \pi^{i-1}_{p_i}, \text{ and} \\ \pi^i_j &= \pi^{i-1}_j \text{ for all } j \in \{1, \dots, n\} \setminus \{p_i, p_i+1\}. \end{aligned}$$

Here π^u_v denotes the v^{th} element of permutation π^u. We call Π the *list of permutations constructed by applying P to π^0* and denote it by $\Pi(\pi^0, P)$.

Definition 3 (Sequence of swapping points)**.** *An ordered list P of numbers from the set $\{1, 2, \ldots, n-1\}$ is called a* sequence of swapping points *of size n if $\Pi(\pi^0, P)$ is a simple allowable sequence where $\pi^0 = (1, 2, \ldots, n)$.*

If there is no confusion concerning n, we simply call P a sequence of swapping points. From the definitions, it can be seen that there is a one-to-one correspondence between sequences of swapping points of size n and simple allowable sequences of permutations of $\{1, 2, \ldots, n\}$. Note also that the length of a sequence of swapping points is the number of permutations in a simple allowable sequence except the initial permutation and it is given by $\binom{n}{2}$ (54, page 92).

We now give an example of a simple allowable sequence and its corresponding full sequence of swapping points.

Example 2. *A simple allowable sequence with the two elements involved in each swap shown above the arrow:*

$$\begin{aligned}(1,2,3,4,5) &\xrightarrow{3,4} (1,2,4,3,5) \xrightarrow{1,2} (2,1,4,3,5) \xrightarrow{1,4} (2,4,1,3,5) \xrightarrow{2,4} \\ (4,2,1,3,5) &\xrightarrow{3,5} (4,2,1,5,3) \xrightarrow{1,5} (4,2,5,1,3) \xrightarrow{1,3} (4,2,5,3,1) \xrightarrow{2,5} \\ &(4,5,2,3,1) \xrightarrow{4,5} (5,4,2,3,1) \xrightarrow{2,3} (5,4,3,2,1)\end{aligned}$$

The corresponding sequence of swapping points is $(3, 1, 2, 1, 4, 3, 4, 2, 1, 3)$.

We conclude this session by noting the following consequence of the uniqueness property.

Lemma 6 (Simple uniqueness property)**.** *Any two consecutive elements in a sequence of swapping points must be different.*

Proof. We prove the lemma by contradiction. Assume that there is a sequence of swapping points of size n with the form $P = (\ldots, i, i, \ldots), i \in \{1, \ldots, n-1\}$. Apply P to the identity permutation π^0. Let $\pi = (\pi_1, \pi_2, \ldots, \pi_i, \pi_{i+1}, \ldots, \pi_n)$ be the permutations obtained just before applying the portion (i, i) of P. It is obvious that the pair $\{\pi_i, \pi_{i+1}\}$ will be swapped twice. This contradicts the uniqueness property. $\square$

6.2.2 Representing a sequence of permutations: The wiring diagram

In this section we describe how to construct a *wiring diagram* (15; 54) to represent an allowable sequence of permutations. We start with a simple allowable sequence of permutations: Consider a simple allowable sequence Π of size n and its corresponding full sequence of swapping points $P = (p_1, p_2, \ldots, p_k)$. Let n horizontal "wires", labelled from 1 to n from top to bottom, going from left to right. For each element p_i of the sequence of swapping points, let the wire at position p_i intersect with the wire at position $p_i + 1$ (the positions are identified top-down) once, then both continue to go right. Note that after the intersection, the two wires swap their positions but keep their labels. Doing this with all swapping points results in a diagram illustrating the given simple allowable sequence. Figure 6.1 shows the wiring diagram representing the simple allowable sequence given in Example 2.

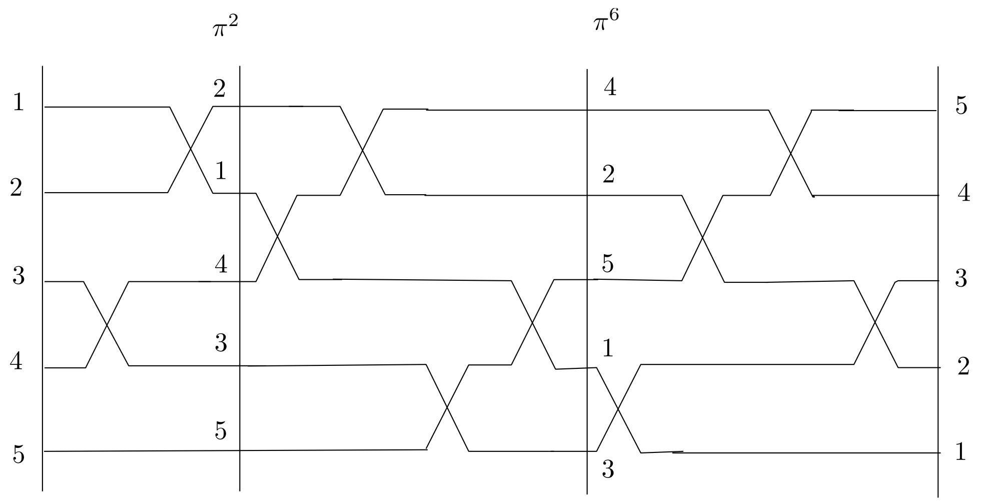


Figure 6.1: The wiring diagram that represents the simple allowable sequence in Example 2.

Returning the simple allowable sequence of permutations from the wiring diagram is straightforward: After the initial identity permutation, simply read out the labels of the wires from top to bottom after the intersection concerning swapping point p_i to get permutation π^i. For example, the two vertical lines in the diagram in Figure 6.1 give two the permutations: $\pi^2 = (2, 1, 4, 3, 5)$ and $\pi^6 = (4, 2, 5, 1, 3)$.

A wiring diagram can also be employed to represent a general (not necessarily simple) allowable sequence of permutations in the same fashion. The only difference is that each intersection, instead of involving 2 wires, now involves all the wires that correspond to the elements taking part in the reversal. The permutations can be read out from the diagram in the same way as with the simple allowable sequence. Figure 6.2 shows the wiring diagram of the allowable sequence given in Example 1.

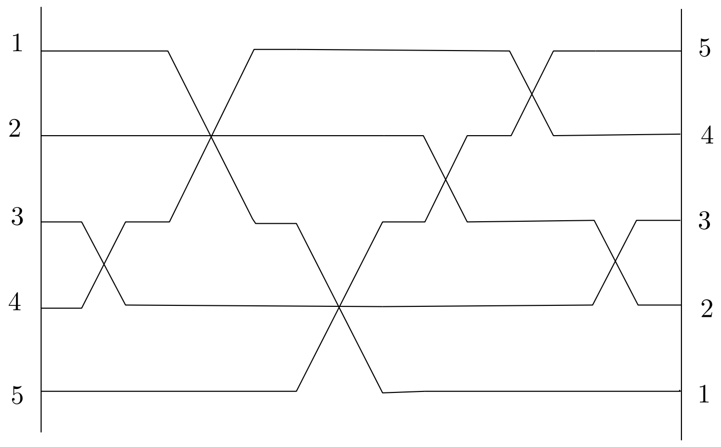


Figure 6.2: The wiring diagram for the (non-simple) allowable sequence given in Example 1.

6.3 What is a FSS?

In this section we first combine the initial permutation and the sequence of swapping points to define the so called p-sequence. We then use this concept to define FSSs, and show how to use wiring diagrams to illustrate such split systems. After that we will briefly explain why FSSs are called "flat" and indicate how to visualise such split systems using planar split networks. Finally we show that affine split systems are a subclass of the FSSs.

6.3.1 Definition of FSSs

FSSs are defined using the following concept.

Definition 4 ($p-$sequence)**.** *Given a set $X = \{1, 2, \ldots, n, \}$, a $p-$sequence on X is a pair $\mathcal{P} = (\pi^0, P)$ where π^0 is an arbitrary permutation of X and $P = (p_1, p_2, \ldots, p_k)$ is a sequence of swapping points of size n.*

From a $p-$sequence $\mathcal{P}$, we derive a split system $\mathfrak{S}(\mathcal{P}) = \{S_1, S_2, \ldots, S_k\}$ where $S_i = \{\pi^i_1, \ldots, \pi^i_{p_i}\}|\{\pi^i_{p_i+1}, \ldots, \pi^i_n\}$ for all $i \in \{1, \ldots, k\}$, and $\pi^i = (\pi^i_1, \pi^i_2, \ldots, \pi^i_n)$ is the i^{th} permutation in the simple allowable sequence constructed by applying P to π^0.

Definition 5 (Flat split system)**.** *Given a set $X = \{1, 2, \ldots, n\}$, a split system $\mathfrak{S}$ on X is called a* Flat Split System (FSS) *if and only if there exists a $p-$sequence $\mathcal{P}$ on X such that $\mathfrak{S} \subseteq \mathfrak{S}(\mathcal{P})$. If $\mathfrak{S} = \mathfrak{S}(\mathcal{P})$ then it is called a* full FSS*.*

The FSS was mentioned before under the name *pseudo-affine split system* in Bryant and Dress's study about so-called *linearly independent split systems* (22). In that paper it was also remarked that the class of FSSs is a superclass of affine split systems. We will prove this in more detail using another approach later on. We will also show that FSSs can be visualised by planar split networks - the fact that give them their name "Flat Split Systems". We now study some basic properties of the FSS. From the definition, we immediately have:

Theorem 12. *A FSS on n taxa can have at most $\begin{pmatrix} n \\ 2 \end{pmatrix}$ splits. Furthermore, it has $\begin{pmatrix} n \\ 2 \end{pmatrix}$ splits if and only if it is full.*

Proof. An FSS can have no more splits than the full FSS containing it. The number of splits in a full FSS is the length of the sequence of swapping points in the corresponding p-sequence. As noted above, the length of a sequence of swapping points is $\begin{pmatrix} n \\ 2 \end{pmatrix}$. □

Next we explain the relationship between a FSS and the wiring diagram. First, getting a FSS from a wiring diagram is straightforward: For each swapping point, the labels of the wires above and below the after intersection, respectively, form the two blocks of the corresponding split. For illustration, consider the wiring diagram in Figure 6.1: After the first intersection (between wires 3 and 4), we have split $\{1,2,4\}|\{3,5\}$, after the second intersection (between wires 1 and 2) we have split $\{2\}|\{1,4,3,5\}$, after the third intersection (between wires 1 and 4) we have split $\{2,4\}|\{1,3,5\}$, and so on. Finally the full FSS with splits written in the order corresponding to that in the sequence of swapping points is

$$\big\{\{1,2,4\}|\{3,5\}, \{2\}|\{1,4,3,5\}, \{2,4\}|\{1,3,5\}, \{4\}|\{2,1,3,5\}, \{4,2,1,5\}|\{3\},$$
$$\{4,2,5\}|\{1,3\}, \{4,2,5,3\}|\{1\}, \{4,5\}|\{2,3,1\}, \{5\}|\{4,2,3,1\}, \text{ and } \{5,4,3\}|\{2,1\}.\big\}$$

If we keep the diagram and change the labels of the wires, we may obtain a different full FSS using the same method of reading out the splits as above. For example, re-labelling the wires in Figure 6.1 by $(2,5,1,4,3)$ from top to bottom results in the wiring diagram depicted in Figure 6.3 and a different corresponding FSS:

$$\big\{\{2,5,4\}|\{1,3\}, \{5\}|\{2,4,1,3\}, \{5,4\}|\{2,1,3\}, \{4\}|\{5,2,1,3\}, \{4,5,2,3\}|\{1\},$$
$$\{4,5,3\}|\{2,1\}, \{4,5,3,1\}|\{2\}, \{4,3\}|\{5,1,2\}, \{3\}|\{4,5,1,2\}, \{3,4,1\}|\{5,2\}.\big\}$$

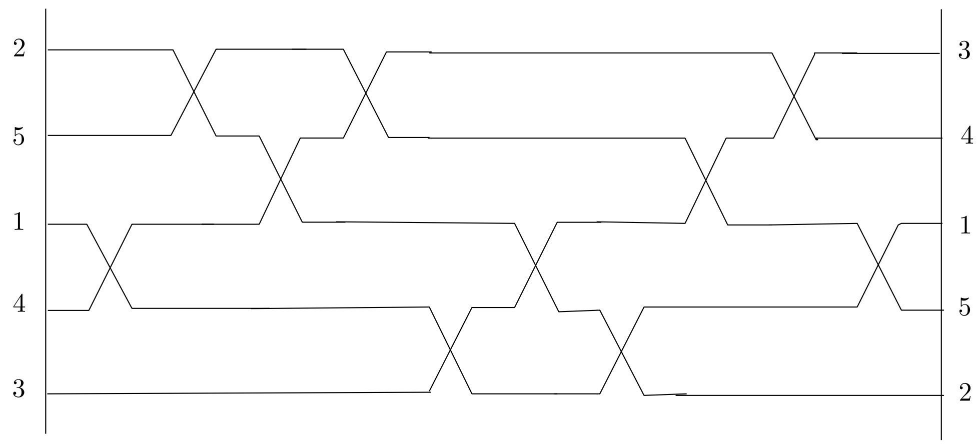


Figure 6.3: A wiring diagram with the same shape as in Figure 6.1 but a different labelling of the wires.

The examples above also demonstrate how to create a labelled wiring diagram to represent a FSS if the FSS is given in the form of a p-sequence $\mathcal{P} = (\pi^0, P)$.

To do this, first construct a wiring diagram for the sequence of swapping points P. Then re-label the wires from top to bottom by the elements of the initial permutation π^0. We call P the *topology* and π^0 the *labelling* of the wiring diagram and also of the p-sequence $\mathcal{P}$. If π^0 is the identity permutation, then we say the wiring diagram/p-sequence has a *standard labelling.*

The analysis above is applied to full FSSs. However, from the definition, each (non-full) FSS is always a subset of a full FSS, therefore there is always a wiring diagram that "accommodates" the splits of the former. From a practical point of view, every FSS can be considered full by adding some splits to make it full and assigning weight 0 to these additional splits.

6.3.2 Visualising a FSS by a planar split network

In this section we indicate how a FSS can be visualised by a planar split system, which in turn explains why the name "flat split system" is selected. Our method is a combination of an algorithm for computing the dual of a so-called *arrangement of pseudolines* (4; 54) and an algorithm for computing a so-called *zonotopal tiling* to represent an allowable sequence (see e.g. (54, p. 100-105)). The term pseudoline was coined by Levi (95) as a curve whose intersection behavior to some other pseudolines is similar to that between straight lines, i.e. two pseudolines have at most one common point and if such point exists, the two pseudolines cross each others at this point. An arrangement of pseudolines is a family of pseudolines where each pseudoline intersects with every other one. Note that a wiring diagram is a special case of an arrangement of pseudolines. The explanation on how to represent a FSS using a planar split network will be done through an example but this can be easily generalised.

Given a (not necessarily full) FSS $\mathfrak{S}$ with weight vector ω. Let $\mathcal{P} = (\pi, P)$ where $\pi = (x_1, x_2, \ldots, x_n), P = (p_1, p_2, \ldots, p_k), k = \begin{pmatrix} n \\ 2 \end{pmatrix}$ be a $p-$sequence such that $\mathfrak{S} \subseteq \mathfrak{S}(\mathcal{P})$. Figure 6.4 shows in form of a wiring diagram an example where $\pi = (x_1, x_2, x_3, x_4, x_5), P = (1, 3, 2, 4, 3, 1, 2, 4, 1, 3)$, and the shaded faces in the wiring diagram correspond to the splits of the system to be drawn. The splits are read from left to right with taxa laying above the intersections written in the left blocks. This results in the ordered splits shown in Figure 6.5(a). In general, let $S_1, S_2, \ldots, S_m$ be the splits in $\mathfrak{S}$ sorted in order in which they are induced by $\mathcal{P}$. For each split $S_i, i = 1, \ldots, m$, let L_i be the set of taxa in the left block.

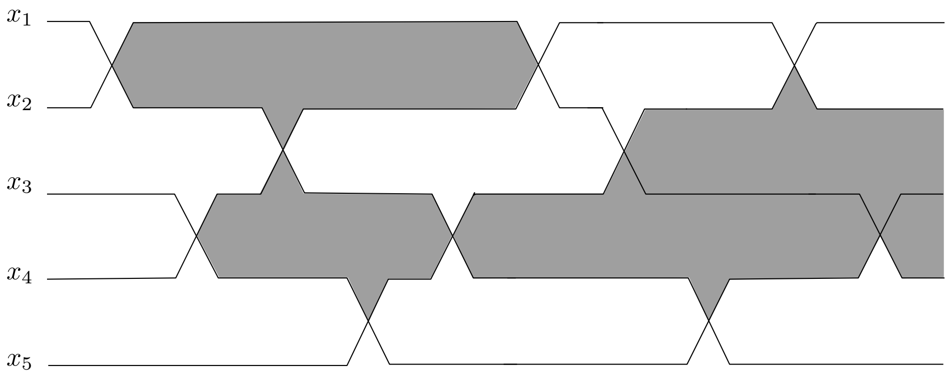
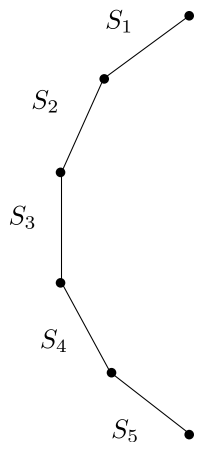


Figure 6.4: A wiring diagram whose corresponding full FSS contains the splits (shaded) of the FSS to be drawn.

$$S_1 = \{x_2\}|\{x_1, x_3, x_4, x_5\}$$
$$S_2 = \{x_1, x_2, x_4\}|\{x_3, x_5\}$$
$$S_3 = \{x_2, x_4, x_5\}|\{x_1, x_3\}$$
$$S_4 = \{x_4, x_5\}|\{x_1, x_2, x_3\}$$
$$S_5 = \{x_3, x_4, x_5\}|\{x_1, x_2\}$$

(a)



(b)

Figure 6.5: Visualising a FSS - the starting point: (a) The (ordered) splits to be drawn. (b) The initial split network.

The drawing always starts with the network as a simple path as shown in Figure 6.5(b) where the edges, reading from top to bottom, correspond to the ordered splits $S_1, S_2, \ldots, S_m$. The lengths of the edges are set as the weights of the corresponding splits. We now process the taxa in order in which they occur in π. For the i^{th} taxon, $1 \leq i \leq n$, we re-order the splits (if necessary) so that the following property is held.

Property 4 (Separation)**.** *There exists a unique* $t \in \{1, \ldots, m\}$ *such that for all* $1 \leq j < t, x_i \in L_j$ *and for all* $t \leq j \leq m, x_i \notin L_j$.

The re-ordering is made by swapping the order of two consecutive splits. After a finite number of swaps, Property 4 must hold. Every time two splits are swapped, we adjust the network accordingly by making a parallelogram starting with two rightmost edges corresponding to the two involving splits - the parallelogram is uniquely identified. Once all the swaps are made and parallelograms are added, look at the vertices on the right boundary of the network, label the t^{th} vertex (from top to bottom), where t is the unique index identified by the separation property, by x_i.

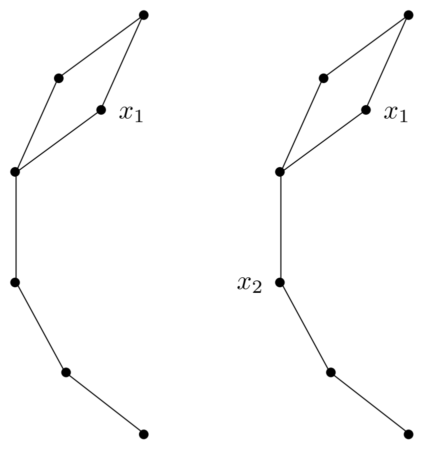
In our example, for x_1, it is clear that we only need to swap S_1 and S_2. The new order of the splits is shown in Figure 6.6(a) with $t = 2$. The updated network with the parallelogram added and x_1 placed is shown in Figure 6.6(b). For x_2, the current order of splits satisfies the separation property already with $t = 4$. We place x_2 at the fourth vertex on the right boundary and show the new network in Figure 6.6(c).

When consider x_3, we have a case where more than one swaps are necessary. In particular S_5 is to be swapped with four splits S_4, S_3, S_1 and S_2 in that order. The new order of splits is shown in Figure 6.7(a) with $t = 2$. The four new parallelograms and the position of x_3 are updated in Figure 6.7(b). Continue with x_4 and x_5 in the analogous way, we have the final network in Figure 6.7(c). A more careful study will show that usually the resulting network can be simplified in the sense of having less edges. Figure 6.7(d) illustrates a network that is equivalent to that in Figure 6.7(c) but with less edges (11 comparing to 23). It is easy to check that both the networks display all the required splits and nothing else.

We note that the fact that by re-ordering we can always modify the order of the splits so that the separation property holds relies on the uniqueness property, i.e. any two taxa in a p-sequence swap their positions at most once. As a consequence, any two splits swap their positions during the construction at most once,

$$\begin{aligned}
S_2 &= \{x_1, x_2, x_4\}|\{x_3, x_5\}\\
S_1 &= \{x_2\}|\{x_1, x_3, x_4, x_5\}\\
S_3 &= \{x_2, x_4, x_5\}|\{x_1, x_3\}\\
S_4 &= \{x_4, x_5\}|\{x_1, x_2, x_3\}\\
S_5 &= \{x_3, x_4, x_5\}|\{x_1, x_2\}
\end{aligned}$$

(a)



(b) (c)

Figure 6.6: Visualising a FSS - the first steps: (a) The new order of the splits after processing x_1. (b) The network with x_1 fixed. (c) The network with x_2 fixed.

and therefore we can always add a suitable parallelogram without introducing crossing edges to the network. There are still several issues to discuss on the visualisation work, namely how to compute the coordinates of the vertices and the angles between pairs of consecutive edges on the initial path, how to simplify the resulting network, and the complexity of the method. All these will be presented in detail in (135). In the next section we will show that FSSs are a superclass of affine split systems.

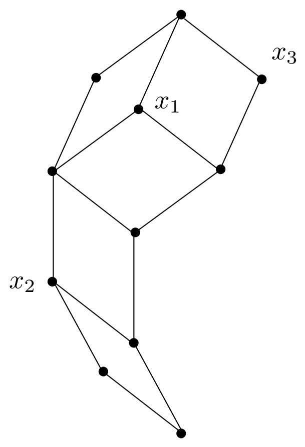
6.3.3 Affine split systems are a subclass of the FSSs

We have shown before that circular split systems are a subclass of the class of affine split systems, see Section 5.3. Now we will prove that affine split systems are a subclass of the class of FSSs. We will follow the basic idea behind the range-counting algorithm presented in Section 15.7 of the book by Edelsbrunner (41). A similar idea was also used by Welzl to compute the visibility graph of a set of straight line segments on the plane (148).

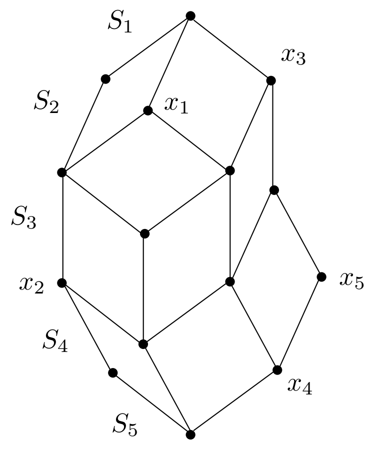
Consider an arbitrary affine split system Σ on X. Let Q denote a set of n points that represents the elements in X. We assume that no two points in Q

$$S_5 = \{x_3, x_4, x_5\}|\{x_1, x_2\}$$
$$S_2 = \{x_1, x_2, x_4\}|\{x_3, x_5\}$$
$$S_1 = \{x_2\}|\{x_1, x_3, x_4, x_5\}$$
$$S_3 = \{x_2, x_4, x_5\}|\{x_1, x_3\}$$
$$S_4 = \{x_4, x_5\}|\{x_1, x_2, x_3\}$$

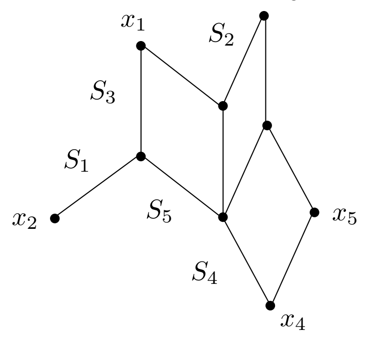
(a)



(b)



(c)



(d)

Figure 6.7: Visualising a FSS - the final steps: (a) The new order of the splits after processing x_3. (b) The updated network with x_3 fixed. (c) The final split network. (d) The final split network simplified.

have the same x-coordinate (otherwise we can rotate Q slightly). The points in Q can also be chosen so that no three points are co-linear.

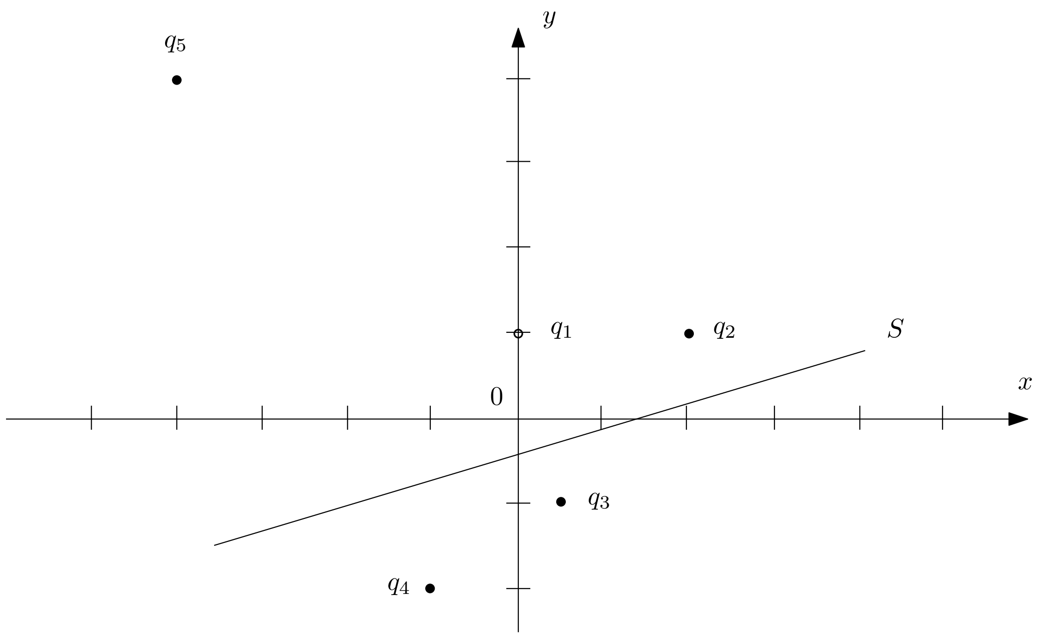
First we map the points in Q to straight lines on the "dual plane", given by

$$q = (a, b) \mapsto q^* = \{(x, y) \in \mathbb{R}^2 : y = ax - b\}.$$

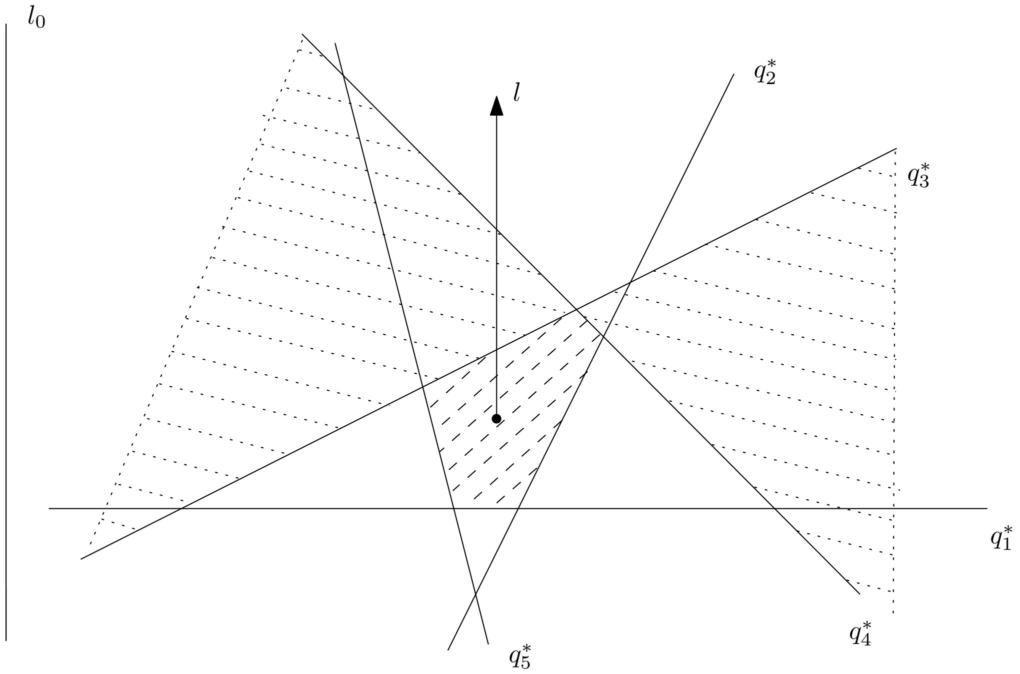
Let $\mathcal{L} = \{q^* : q \in Q\}$. The lines in $\mathcal{L}$ partition the dual plane into a family $\mathcal{C}$ of convex regions whose leftmost points are the intersections of the lines. The assumption that no two points have the same x-coordinate ensures that every two lines intersect with each other. The assumption that no three points are co-linear ensures that every point in the plane is contained in at most two lines in $\mathcal{L}$. These assumptions imply $|\mathcal{C}| = \begin{pmatrix} n \\ 2 \end{pmatrix}$. Note that the convex regions in $\mathcal{C}$ are in one-to-one correspondence with the affine splits of the point set Q. Moreover, the set of affine splits that separate two points $q, r \in Q, q \neq r$ corresponds to the set of convex regions in $\mathcal{C}$ that are contained in the double-wedge bounded by q^* and r^*.

In Figure 6.8 we give an example: Figure 6.8(a) shows a set Q of five points with split S as a line separating $\{q_3, q_4\}$ from $\{q_1, q_2, q_5\}$. Figure 6.8(b) shows the set of lines transformed from the points in Q with the shaded convex region corresponding to split S. To get the split, starting at any point inside the convex region, draw a vertical line l upward. Those points $q \in Q$ whose corresponding lines $q^* \in \mathcal{L}$ intersect with l belong to the same block of the split. The remaining points form the other block. In Figure 6.8(b), the sparsely dotted regions also show the double-wedge bounded by q_3^* and q_4^*. It can be checked that the convex regions in the double-wedge correspond to all splits that separate q_3 and q_4. They are: $\{q_3\}|\{q_1, q_2, q_4, q_5\}$, $\{q_2, q_3\}|\{q_1, q_4, q_5\}$, $\{q_4\}|\{q_1, q_2, q_3, q_5\}$, and $\{q_4, q_5\}|\{q_1, q_2, q_3\}$.

Next we will form a p-sequence $\mathcal{P} = (\pi^0, P)$ corresponding to a FSS that contains the given affine split systems. We first label the lines in $\mathcal{L}$ with the indices of the corresponding elements in X, then record the lines' labels at $-\infty$ from top to bottom to form the initial permutation π^0. After that we sweep a vertical line l_0 from $-\infty$ to $+\infty$. Every time l_0 goes across an intersection between $q^*, r^* \in \mathcal{L}$, record the position of q^* or r^* in the permutation, whichever is *smaller*,



(a)



(b)

Figure 6.8: From an affine split system to a FSS - The dual plane approach.

to get a swapping point. This is followed by the interchange of positions of q^* and r^* in the permutation. Once l_0 passes the right-most intersection, we have the sequence of swapping points P.

For illustration, consider the example in Figure 6.8(b): Clearly the initial permutation is $\pi^0 = (5, 4, 1, 3, 2)$. The first intersection that l_0 goes through is that between q_1^* and q_3^*, the positions of q_3^* is 4 and that of q_1^* is 3, so we record 3 as the first swapping point, the permutation becomes $(5, 4, 3, 1, 2)$. Next l_0 goes across the intersection between q_4^* and q_5^* resulting in swapping point 1 and new permutation $(4, 5, 3, 1, 2)$, and so on. At the end the sequence of swapping points is $P = (3, 1, 2, 3, 4, 3, 1, 2, 1, 3)$; the corresponding full FSS is $\mathfrak{S} = \{\{q_5, q_4, q_3\}|\{q_1, q_2\}, \{q_4\}|\{q_5, q_3, q_1, q_2\}, \{q_4, q_3\}|\{q_5, q_1, q_2\}, \{q_4, q_3, q_1\}|\{q_5, q_2\}, \{q_4, q_3, q_1, q_2\}|\{q_5\}, \{q_4, q_3, q_2\}|\{q_1, q_5\}, \{q_3\}|\{q_4, q_2, q_1, q_5\}, \{q_3, q_2\}|\{q_4, q_1, q_5\}, \{q_2\}|\{q_3, q_4, q_1, q_5\}, \{q_2, q_3, q_1\}|\{q_4, q_5\}\}$. It can be checked that $\mathfrak{S}$ contains all possible splits in Figure 6.8(a). In the next section we are going to describe some operations that are used to manipulate the FSSs.

6.4 Operations on FSSs

We know that each p-sequence gives rise to a full FSS. We now define four basic operations on the p-sequences, three of them change the topology and the last one changes the labelling. These operations will be building blocks for manipulating the corresponding FSSs, which will allow us to explore the space of FSSs in the next chapter.

6.4.1 Swapping

Operation `Swapping` creates a new p-sequence by exchanging the values of two consecutive swapping points subject to a validating condition.

Definition 6 (`Swapping`). *Given $\mathcal{P} = (\pi^0, (p_1, \ldots, p_k))$, for any $i \in \{1, \ldots, k-1\}$ such that $|p_i - p_{i+1}| \geq 2$, p_i and p_{i+1} are swapped to make a new p-sequence $\mathcal{P}' = (\pi^0, (p_1, \ldots, p_{i-1}, p_{i+1}, p_i, p_{i+2}, \ldots, p_k))$. The operation is denoted as $Sw(\mathcal{P}, i)$.*

We first show that this operation always yields a valid p-sequence. Clearly it is sufficient to show the following.

Lemma 7. *A valid sequence of swapping points cannot have a portion of the form (v, u, v) where $|u - v| \geq 2$.*

Proof. Assume that there is a sequence of swapping points $P = (\dots, p_i = v, u, v, \dots)$ where $|u - v| \geq 2$. Assume that $u < v$, the proof for the case $u > v$ is similar. Apply P on an arbitrary permutation. Let $(\dots, \pi_u, \pi_{u+1}, \dots, \pi_v, \pi_{v+1}, \dots)$ be the permutation just before applying p_i. Then the permutations obtained by applying v, u, v are:

$$\begin{aligned}
\pi^i &= (\dots, \pi_{\mathbf{u+1}}, \pi_{\mathbf{u}}, \dots, \pi_v, \pi_{v+1}, \dots) \\
\pi^{i+1} &= (\dots, \pi_{u+1}, \pi_u, \dots, \pi_{\mathbf{v+1}}, \pi_{\mathbf{v}}, \dots) \\
\pi^{i+2} &= (\dots, \pi_{\mathbf{u}}, \pi_{\mathbf{u+1}}, \dots, \pi_{v+1}, \pi_v, \dots).
\end{aligned}$$

But then the pair $\{\pi_u, \pi_{u+1}\}$ is swapped twice in π^i and π^{i+2}. This contradicts the uniqueness property. □

Having proved Lemma 7, we know that `Swapping` always produces a valid p-sequence. We now prove that the operation does not alter the corresponding FSS:

Lemma 8. *`Swapping` operation does not change the corresponding FSSs.*

Proof. Let $\mathfrak{S}(\mathcal{P}) = \{S_1, S_2, \dots, S_k\}$ and $\mathfrak{S}(\mathcal{P}') = \{S'_1, S'_2, \dots, S'_k\}$. We have to prove that $\mathfrak{S}(\mathcal{P}) = \mathfrak{S}(\mathcal{P}')$. It is obvious that $S_j = S'_j$ for all $j \in \{1, \dots, i-1\}$. Let $u = p_i, v = p_{i+1}$. We assume that $u < v$, the case where $u > v$ can be checked similarly.

Let $\pi^{(i-1)} = \{\dots, \pi_u, \pi_{u+1}, \dots, \pi_v, \pi_{v+1}, \dots\}$ be the $(i-1)^{th}$ permutation in the simple allowable sequence constructed by applying P to π^0. It is the same as the $(i-1)^{th}$ permutation $\pi'^{(i-1)}$ in the simple allowable sequence constructed by applying P' to π^0. Then we have:

$$\begin{aligned}
\pi^i &= (\dots, \pi_{\mathbf{u+1}}, \pi_{\mathbf{u}}, \dots, \pi_v, \pi_{v+1}, \dots) \\
\pi^{(i+1)} &= (\dots, \pi_{u+1}, \pi_u, \dots, \pi_{\mathbf{v+1}}, \pi_{\mathbf{v}}, \dots)
\end{aligned}$$

and

$$\begin{aligned}
\pi'^i &= (\dots, \pi_u, \pi_{u+1}, \dots, \pi_{\mathbf{v+1}}, \pi_{\mathbf{v}}, \dots,) \\
\pi'^{(i+1)} &= (\dots, \pi_{\mathbf{u+1}}, \pi_{\mathbf{u}}, \dots, \pi_{v+1}, \pi_v, \dots).
\end{aligned}$$

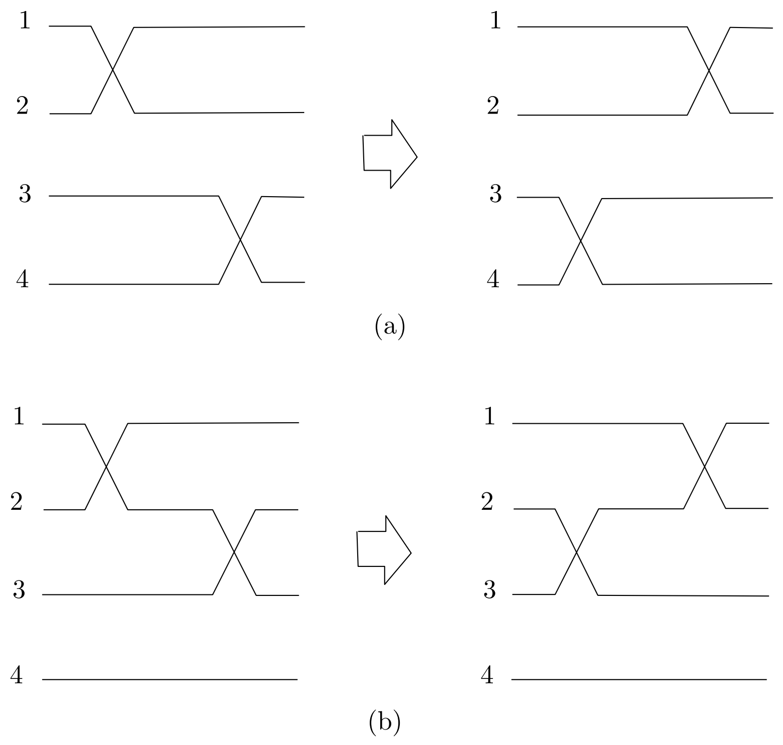


Figure 6.9: `Swapping` operation: (a) A valid case. (b) An invalid case.

The condition $|u - v| \geq 2$ is needed to ensure that the swaps involving (π_u, π_{u+1}) and (π_v, π_{v+1}) are independent of each other. As the result, we have:

$$\begin{aligned} S_i &= \{\dots, \pi_{\mathbf{u+1}}\}|\{\pi_{\mathbf{u}}, \dots, \pi_v, \pi_{v+1}, \dots\} \\ S_{(i+1)} &= \{\dots, \pi_{u+1}, \pi_u, \dots, \pi_{\mathbf{v+1}}\}|\{\pi_{\mathbf{v}}, \dots\} \end{aligned}$$

and

$$\begin{aligned} S'_i &= \{\dots, \pi_u, \pi_{u+1}, \dots, \pi_{\mathbf{v+1}}\}|\{\pi_{\mathbf{v}}, \dots\} \\ S'_{(i+1)} &= \{\dots, \pi_{\mathbf{u+1}}\}|\{\pi_{\mathbf{u}}, \dots, \pi_{v+1}, \pi_v, \dots\}. \end{aligned}$$

It can be seen that $S_i = S'_{i+1}$ and $S_{i+1} = S'_i$. In addition, $\pi^{(i+1)} = \pi'^{(i+1)}$ and the rest of P and P' are identical. Therefore $S_j = S'_j$ for all $j \in \{i+2, \dots, k\}$. Finally we have $\mathfrak{S}(\mathcal{P}') \equiv \mathfrak{S}(\mathcal{P})$. Hence, $\mathfrak{S}(\mathcal{P}')$ is just $\mathfrak{S}(\mathcal{P})$ with splits i and $i+1$ interchanged. $\square$

Figure 6.9 illustrates the swapping operation. In Figure 6.9(a), the swapping points involved are 1 and 3, the original splits are $S_1 = \{2\}|\{1,3,4\}$ and $S_2 = \{2,1,4\}|\{3\}$. When the swapping points are interchanged, the corresponding splits are $S'_1 = \{1,2,4\}|\{3\}$ and $S'_2 = \{2\}|\{1,4,3\}$. It is obvious that $S_1 = S'_2, S_2 = S'_1$. In Figure 6.9(b), the swapping points are 1 and 2. The splits before the interchange are $\{2\}|\{1,3,4\}$ and $\{2,3\}|\{1,4\}$, and those after that are $\{1,3\}|\{2,4\}$ and $\{3\}|\{1,2,4\}$. It is evident that the two resulting split systems are different. This is because the swapping points do not satisfy the condition $|p_i - p_{i+1}| \geq 2$.

6.4.2 Flipping

From Lemma 7 we know that if there is a portion of the form (u, v, u) in a sequence of swapping points then $|u - v| < 2$. By Lemma 6 we also know that there can't be two consecutive swapping points with the same value. So if the portion (u, v, u) exists then v can only be either $u+1$ or $u-1$. We call such a triple *viable*. `Flipping` operation works on viable triples.

Definition 7 (`Flipping`). *Given $\mathcal{P} = (\pi^0, P)$. If $P = (\dots, u, u+1, u, \dots)$ then a new p-sequence can be formed as $\mathcal{P}' = (\pi^0, P')$ where $P' = (\dots, u+1, u, u+1 \dots)$.*

This is called positive flipping*. Similarly, if* $P = (\dots, u, u-1, u, \dots)$ *then* $\mathcal{P}' = (\pi^0, P')$ *where* $P' = (\dots, u-1, u, u-1, \dots)$*. This is called* negative flipping*. The operation is denoted as* $Fl(\mathcal{P}, i)$ *where* i *is the index of the first value in the viable triple.*

As with the swapping operation, we prove that changing the values of swapping points in `Flipping` does not invalidate the resulting sequence of swapping points.

Lemma 9. *A valid sequence of swapping points* P *cannot have a portion of the form* $(u+1, u, u+1, u)$ *or* $(u, u+1, u, u+1)$*.*

Proof. We only need to show that P cannot contain $(u+1, u, u+1, u)$. The other case can be checked in an analogous way. Assume for a contradiction that there exists a sequence of swapping points $P = (\dots, p_i = u+1, u, u+1, u, \dots)$. Apply P on an arbitrary permutation. Let $(\dots, \pi_u, \pi_{u+1}, \pi_{u+2}, \dots)$ be the permutation just before applying p_i. Then the permutations obtained by applying $u+1, u, u+1, u$ are:

$$\begin{aligned}
\pi^i &= (\dots, \pi_u, \pi_{\mathbf{u+2}}, \pi_{\mathbf{u+1}}, \dots) \\
\pi^{i+1} &= (\dots, \pi_{\mathbf{u+2}}, \pi_{\mathbf{u}}, \pi_{u+1}, \dots) \\
\pi^{i+2} &= (\dots, \pi_{u+2}, \pi_{\mathbf{u+1}}, \pi_{\mathbf{u}}, \dots) \\
\pi^{i+3} &= (\dots, \pi_{\mathbf{u+1}}, \pi_{\mathbf{u+2}}, \pi_u, \dots).
\end{aligned}$$

But the pair $\{\pi_{u+1}, \pi_{u+2}\}$ is swapped twice (in π^i and π^{i+3}). This contradicts the uniqueness property. □

Having proved Lemma 9, we know that the flipping operation always yields a valid p-sequence. We now prove the main characteristic of interest to us for this operation.

Lemma 10. *`Flipping` operation only changes one split in the corresponding FSS.*

Proof. We only consider the positive flipping case. The negative flipping case can be done in a similar way.

Let $\mathfrak{S}(\mathcal{P}) = \{S_1, S_2, \dots, S_k\}$ and $\mathfrak{S}(\mathcal{P}') = \{S'_1, S'_2, \dots, S'_k\}$. Let $i, i+1, i+2$ be the indices of the elements in the triple $(u, u+1, u)$. Then it is obvious that $S_j = S'_j$ for all $j \in \{1, \dots, i-1\}$.

Let $\pi^{(i-1)} = \{\pi_1, \dots, \pi_{u-1}, \pi_u, \pi_{u+1}, \pi_{u+2}, \pi_{u+3}, \dots, \pi_n\}$ be the $(i-1)^{th}$ permutation in the simple allowable sequence constructed by applying P on π^0. It is the same as the $(i-1)^{th}$ permutation $\pi'^{(i-1)}$ in the simple allowable sequence constructed by applying P' on π^0. Then we have:

$$\begin{aligned}
\pi^i &= \{\dots, \pi_{\mathbf{u+1}}, \pi_{\mathbf{u}}, \pi_{u+2}, \dots\} \\
\pi^{(i+1)} &= \{\dots, \pi_{u+1}, \pi_{\mathbf{u+2}}, \pi_{\mathbf{u}}, \dots\} \\
\pi^{(i+2)} &= \{\dots, \pi_{\mathbf{u+2}}, \pi_{\mathbf{u+1}}, \pi_u, \dots\}
\end{aligned}$$

and

$$\begin{aligned}
\pi'^i &= \{\dots, \pi_u, \pi_{\mathbf{u+2}}, \pi_{\mathbf{u+1}}, \dots\} \\
\pi'^{(i+1)} &= \{\dots, \pi_{\mathbf{u+2}}, \pi_{\mathbf{u}}, \pi_{u+1}, \dots\} \\
\pi'^{(i+2)} &= \{\dots, \pi_{u+2}, \pi_{\mathbf{u+1}}, \pi_{\mathbf{u}}, \dots\}.
\end{aligned}$$

Therefore the corresponding splits are:

$$\begin{aligned}
S_i &= \{\dots, \pi_{\mathbf{u+1}}\}|\{\pi_{\mathbf{u}}, \pi_{u+2}, \dots\} \\
S_{(i+1)} &= \{\dots, \pi_{u+1}, \pi_{\mathbf{u+2}}\}|\{\pi_{\mathbf{u}}, \dots\} \\
S_{(i+2)} &= \{\dots, \pi_{\mathbf{u+2}}\}|\{\pi_{\mathbf{u+1}}, \pi_u, \dots\}
\end{aligned}$$

and

$$\begin{aligned}
S'_i &= \{\dots, \pi_u, \pi_{\mathbf{u+2}}\}|\{\pi_{\mathbf{u+1}}, \dots\} \\
S'_{(i+1)} &= \{\dots, \pi_{\mathbf{u+2}}\}|\{\pi_{\mathbf{u}}, \pi_{u+1}, \dots\} \\
S'_{(i+2)} &= \{\dots, \pi_{u+2}, \pi_{\mathbf{u+1}}\}|\{\pi_{\mathbf{u}}, \dots\}.
\end{aligned}$$

It can be seen that $S'_i \neq S_i$ but $S'_{(i+1)} = S_{(i+2)}$ and $S'_{(i+2)} = S_{(i+1)}$. Moreover, $\pi'^{(i+2)} = \pi^{(i+2)}$ and the rest of P and P' are the same. Hence, $S_j = S'_j$ for all $j \in \{i+2, \dots, k\}$. Therefore $\mathfrak{S}(\mathcal{P}')$ only differs from $\mathfrak{S}(\mathcal{P})$ by one split. In more detail, the positive flipping replaces the i^{th} split $\{\dots, \pi_{u+1}\}|\{\pi_u, \pi_{u+2}, \dots\}$ by $\{\dots, \pi_u, \pi_{u+2}\}|\{\pi_{u+1}, \dots\}$ and swaps the order of the $(i+1)^{th}$ and $(i+2)^{th}$ splits. The positive flipping is illustrated in Figure 6.10. $\square$

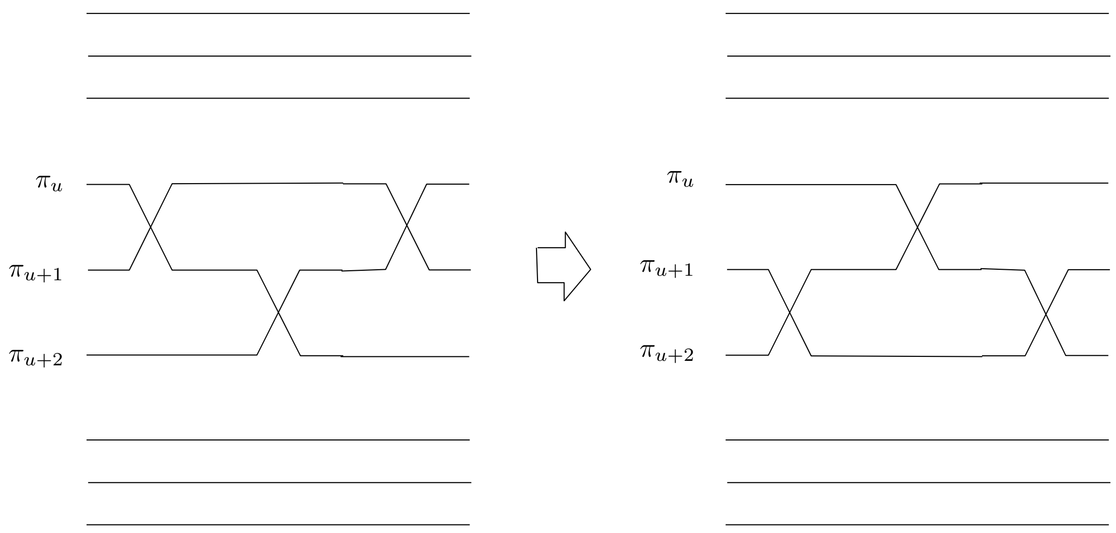


Figure 6.10: An illustration of the positive `Flipping` operation.

It is noticeable that even though operation `Swapping` presented in the previous section does not change the corresponding FSS, it does alter the sequence of swapping points the way that makes `Flipping` applicable, which in turn creates a new FSS. Inspired by the theorem, we combine `Swapping` and `Flipping` to extended the latter as follows.

Definition 8 (`Extended Flipping`)**.** *Given $\mathcal{P} = (\pi^0, P)$. If $P = (\dots, u, B, u+1, C, u, \dots)$ where B, C are (possibly empty) portions of P, neither B or C contains any element of the set $\{u-1, u, u+1\}$, then a new p-sequence can be constructed as $\mathcal{P}' = (\pi^0, P')$ where $P' = (\dots, B, u+1, u, u+1, C \dots)$. This is called* positive ***extended flipping****. Similarly, if $P = (\dots, u, B, u-1, C, u, \dots)$ with the same conditions on B and C, then $\mathcal{P}' = (\pi^0, P')$ where $P' = (\dots, B, u-1, u, u-1, C, \dots)$. This is called* negative ***extended flipping****. We denote this operation as $Fl(\mathcal{P}, i, j, t)$ where $, i, j, t$ are the indices of the involving elements.*

Lemma 11. ***Extended flipping*** *operation only changes one split in the corresponding FSS.*

Proof. On $\mathcal{P}$ we apply `Swapping` $|B|$ times to move the first u to the right of B, then $|C|$ times to move the other u to the left of C, and finally apply the (positive/negative accordingly) standard `Flipping` once to get $\mathcal{P}'$. The negative `Extended flipping` is illustrated in Figure 6.11. □

It is evident that the original flipping operation is a special case of `Extended flipping` where $B = C = \emptyset$. For convenience, from now on, we call the original `Flipping` *standard* whilst when referring to `Flipping` alone, we mean `Extended flipping`. As a consequence, we extend the term *viable triple* to the three elements (not necessarily consecutive) on which the (extended) flipping operation can be applied.

6.4.3 Shifting

The two operations defined so far only change the sequence of swapping points of the input p-sequence. Now we introduce `Shifting` - an operation that alters both the p-sequence's topology and labelling.

Definition 9 (`Shifting`). *Given $\mathcal{P} = (\pi^0, P = (p_1, p_2, \ldots, p_k))$, a new p-sequence can be constructed as $\mathcal{P}' = (\pi^1, P' = (p_2, p_3, \ldots, p_k, n - p_1))$ where π^1 is the second element of the list of permutations constructed by applying P to π^0. The operation is denoted by $Sf(\mathcal{P})$.*

Lemma 12. ***`Shifting`*** *operation does not change the corresponding FSS.*

Proof. Let $\mathfrak{S}(\mathcal{P}) = \{S_1, S_2, \ldots, S_k\}$ and $\mathfrak{S}(\mathcal{P}') = \{S'_1, S'_2, \ldots, S'_k\}$. We have to prove that $\mathfrak{S}(\mathcal{P}) = \mathfrak{S}(\mathcal{P}')$.

Let $\Pi = \{\pi^0, \pi^1, \pi^2, \ldots, \pi^k\}$ be the list of permutations constructed by applying P to π^0. By construction, it is clear that $S'_j = S_{j+1}$ for all $j \in \{1, \ldots, k-1\}$.

Since Π is a simple allowable sequence of permutations, π^k is the reversal of π^0. Let $\pi^0 = (\pi_1, \pi_2, \ldots, \pi_{p_1}, \pi_{p_1+1}, \ldots, \pi_n)$, then $\pi^k = (\pi_n, \ldots, \pi_{p_1+1}, \pi_{p_1}, \ldots, \pi_2, \pi_1)$, and:

$$
\begin{aligned}
S_1 &= \{\pi_1, \pi_2, \ldots, \pi_{p_1+1}\} | \{\pi_{p_1}, \ldots, \pi_n\} \\
S'_k &= \{\pi_n, \ldots, \pi_{p_1}\} | \{\pi_{p_1+1}, \ldots, \pi_2, \pi_1\}.
\end{aligned}
$$

Obviously, $S'_k = S_1$; their blocks just swap positions. So $\mathfrak{S}(\mathcal{P}) = \mathfrak{S}(\mathcal{P}')$. □

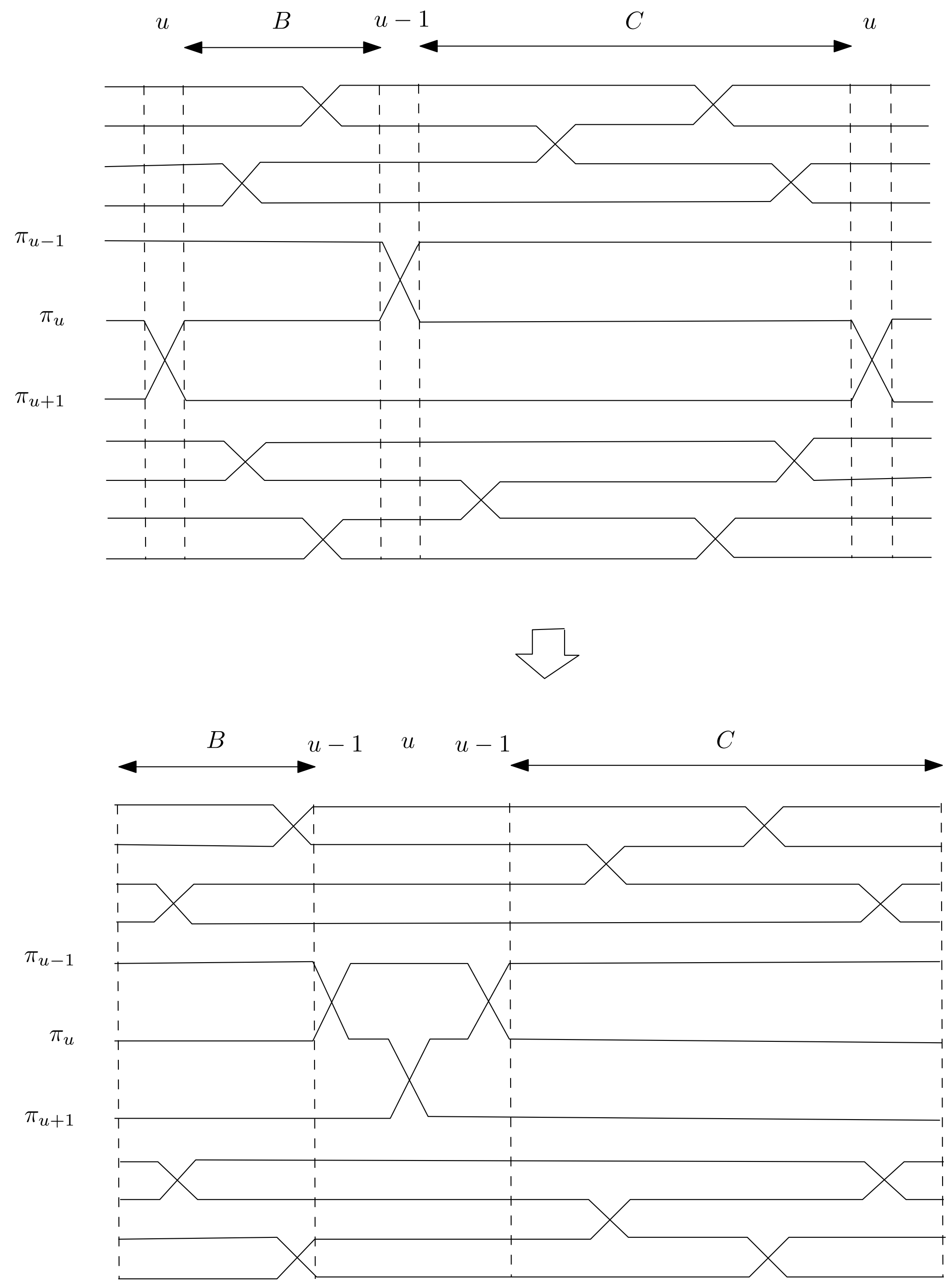


Figure 6.11: An illustration of the negative `Extended flipping` operation.

Figure 6.12 demonstrates `Shifting` operation on a set $X = \{1, 2, 3, 4\}$. In the figure, the diagram bordered by the two solid vertical lines is shifted one position to the diagram bounded by the two dashed lines. The two split systems share all splits represented by the diagram between the first dashed line and the second solid line. It can be checked that the first split of the original FSS is the same as the last split of the "shifted" FSS.

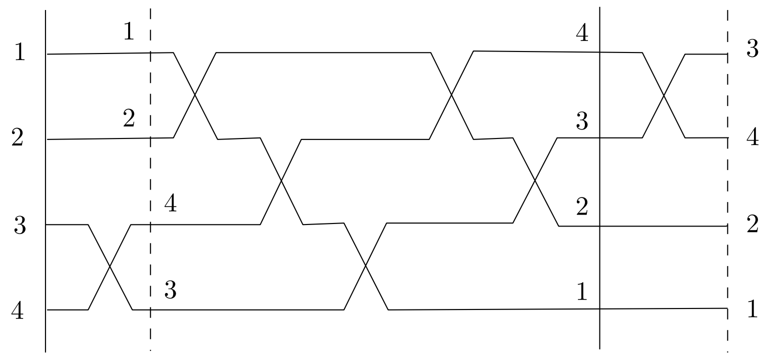


Figure 6.12: The `Shifting` operation.

As with `Flipping`, the shifting operation can be extended so that more than one elements can be shifted at a time. We define the l-`shifting` operation by $\mathcal{P}' = (\pi^l, P' = (p_{l+1}, p_{l+2}, \ldots, p_k, n - p_1, \ldots, n - p_l))$ where π^l is the $(l+1)^{th}$ element of the list of permutations constructed by applying P to π^0. Note that l-`shifting` can clearly be done by continuously applying the shifting operation l times. For convenience, from now on, by referring to the `Shifting` operation, if not stated otherwise, we mean l-`shifting"` for some $l \geq 1$.

The nicest and most important property of `Swapping`, `Flipping` and `Shifting` is that when used together, they allow the exploration of all possible topologies of the p-sequence. This property is made by the following theorem.

Theorem 13 (Ringel (121; 122)). *A sequence of swapping points can be turned to any other one by applying a finite number of* `Swapping`*,* `Flipping` *and* `Shifting` *operations.*

The theorem was introduced in the context of the so-called *simple arrangements of pseudolines* (15; 121; 122) which can easily be converted into sequences of swapping points. The reader can refer to (15, Chapter 6) for an extensive explanation for the transformation.

6.4.4 Permuting

In Section 6.3 it is pointed out that if we fix the topology of a wiring diagram and change the labelling of the wires, we may obtain a different FSS. `Permuting` is an operation that makes use of this observation: It changes the labelling of the p-sequence.

Definition 10 (`Permuting`). *A p-sequence $\mathcal{P} = (\pi, P)$ can be changed to $\mathcal{P}' = (\pi', P)$ where π' is a permutation derived from π.*

There are several strategies to get a new permutation from a given one. However, with reference to Theorem 13, we will require one that allows us to get to any permutation from a given one. In other words, the method must be able to explore all permutations. The combination of such a method with `Swapping` and standard `Flipping` makes it possible to explore the whole space of p-sequences, and therefore FSSs. Let $\pi = (\pi_1, \pi_2, \dots, \pi_n)$, here we summarise some such methods:

- `Exchange`: The positions of two consecutive elements are exchanged, i.e. $(\dots, \pi_i, \pi_{i+1}, \dots) \Rightarrow (\dots, \pi_{i+1}, \pi_i, \dots)$ for some $i \in \{1, \dots, n\}$. If $i = n$ then π_n and π_1 are exchanged. It is known that by a finite number of such interchanges, any permutation can be converted to the identity permutation (86, Problem 5.8). This implies that the operation can get to any permutation from a given one. We denote this type of operation by $Ex(\mathcal{P}, i)$ where i is the index of the first element involved. The exchange strategy is illustrated in Figure 6.13.

- `Pair-Exchange`: This is similar to `Exchange` but does not require the two elements involved be consecutive. We denote this permuting by $Ex(\mathcal{P}, i, j)$ where i, j are the indices of the elements involved, see e.g. Figure 6.14. Being a generalisation of `Exchange`, it obviously allows the exploration of all permutations.

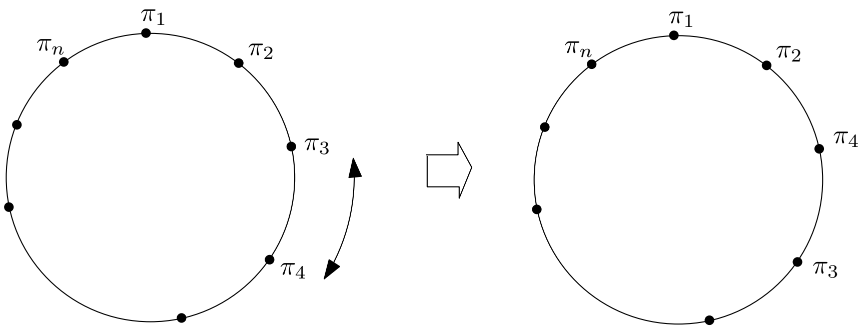


Figure 6.13: `Exchange` permuting strategy.

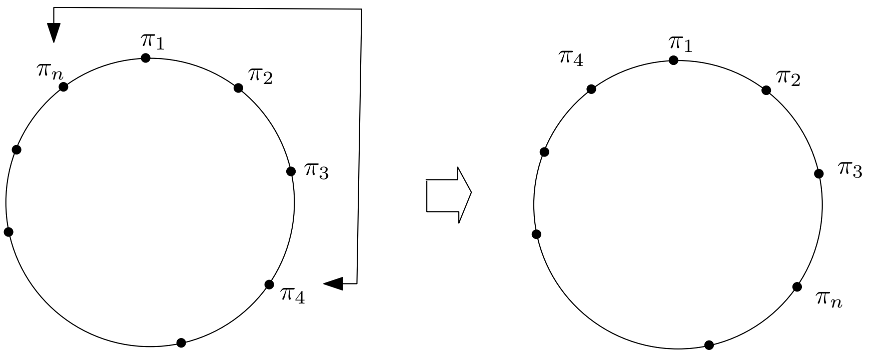


Figure 6.14: `Pair-Exchange` permuting strategy.

- `Reversal`: Place the elements of π in order on a circle. Take a portion of π of length at least 2 and at most l consecutive elements for some $l \in \{2, \ldots, n\}$, then reverse the order of its elements. Denote this operation by $Rev(\mathcal{P}, l, i)$ where l is the length and i is the index of the first element of the portion to be taken. Note that if $l = 2$ then $Rev(\mathcal{P}, 2, i)$ is equivalent to $Ex(\mathcal{P}, i)$. Therefore `Reversal` explores all permutations. An illustration of `Reversal` is shown in Figure 6.15.

- `Prune & Regraft`: Place the elements of π in order on a circle. "Prune" a portion of at most l successive elements for some $l \in \{1, n-1\}$, then "regraft" it to another place. We denote this strategy by $PR(\mathcal{P}, l, i, j)$ where l is the length, i is the index of the first element of the portion and j is the location where the portion to be attached to.

  This operation is motivated by the Sub-tree Pruning and Regrafting (SPR)

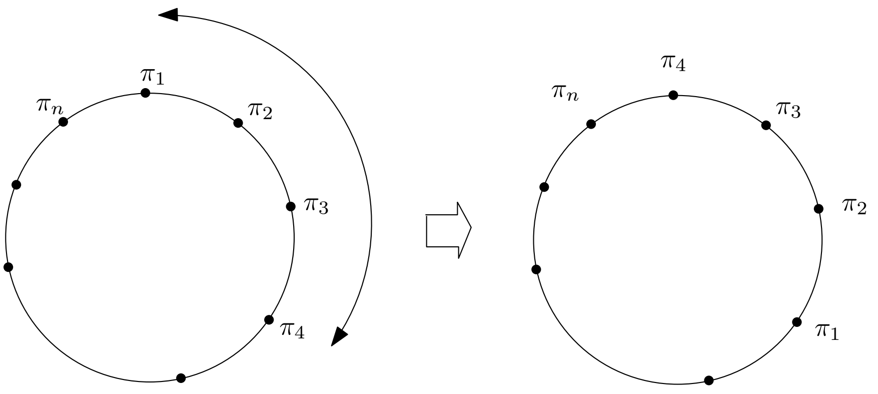


Figure 6.15: `Reversal` permuting strategy.

operation on phylogenetic trees (6; 142) where a subtree is cut off and then regrafted to another position on the tree. It has been proved that a binary phylogenetic tree $\mathfrak{T}$ can be transformed to any other binary phylogenetic tree $\mathfrak{T}'$ using a sequence of SPR operations (17; 140, Proposition 2.6.1). For our `Prune & Regraft`, if $l = 1$ and $j = i + 1$, or $j = 1$ if $i = n$, then it is equivalent to `Exchange`. Thus `Prune & Regraft` satisfies the requirement of being able to explore the whole space of permutations. An illustration of the strategy is presented Figure 6.16.

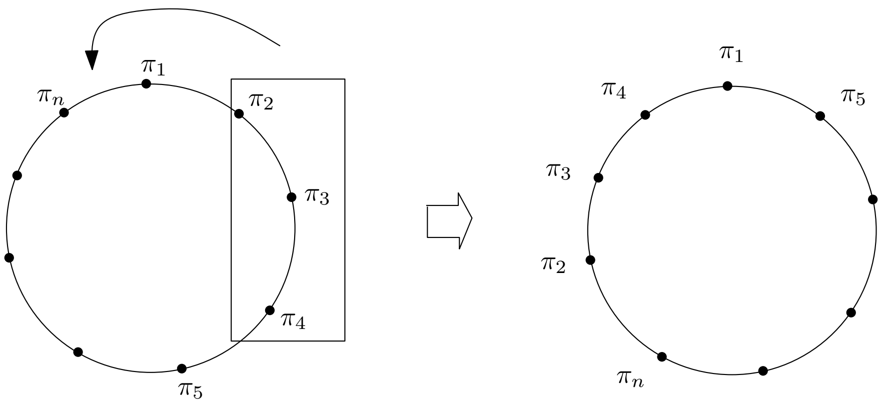
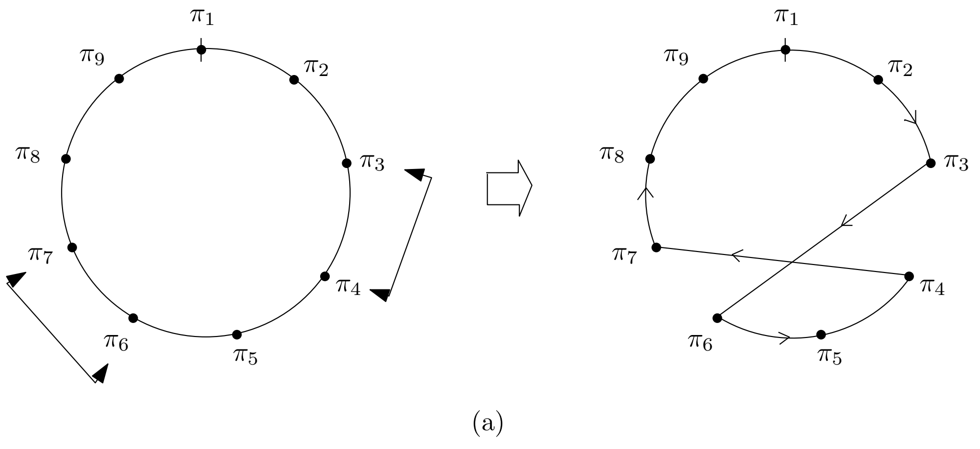


Figure 6.16: `Prune & Regraft` permuting strategy.

- `2-opt move`: For some $i \in \{1, \ldots, n-2\}, j \in \{i+2, \ldots, n\}$, from permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_i, \pi_{i+1}, \ldots, \pi_j, \pi_{j+1}, \ldots, \pi_n)$ we form the new one

$\pi' = (\pi_1, \pi_2, \dots, \pi_i, \pi_j, \pi_{j-1}, \dots, \pi_{i+1}, \pi_{j+1}, \pi_{j+2}, \dots, \pi_n)$ as illustrated in Figure 6.17(a). In the case that $j = n$, we need an additional condition that $i > 1$ and the new permutation is $\pi' = (\pi_1, \pi_2, \dots, \pi_i, \pi_n, \pi_{n-1}, \dots, \pi_{i+1})$, see Figure 6.17(b). We denote this strategy as 2-`opt`$(\mathcal{P}, i, j)$. It can be checked that if $j = i+2$ then 2-`opt`$(\mathcal{P}, i, j)$ is $Ex(\mathcal{P}, i+1)$. As a consequence, 2-`opt` move is valid.

Note that the 2-opt move is a special case of the k-opt move that is used in the context of solving the Travelling Salesman Problem (TSP) heuristically (29; 74). We won't explore the k-opt move where $k > 2$ as it is equivalent to a finite sequence of 2-opt moves (32; 99).



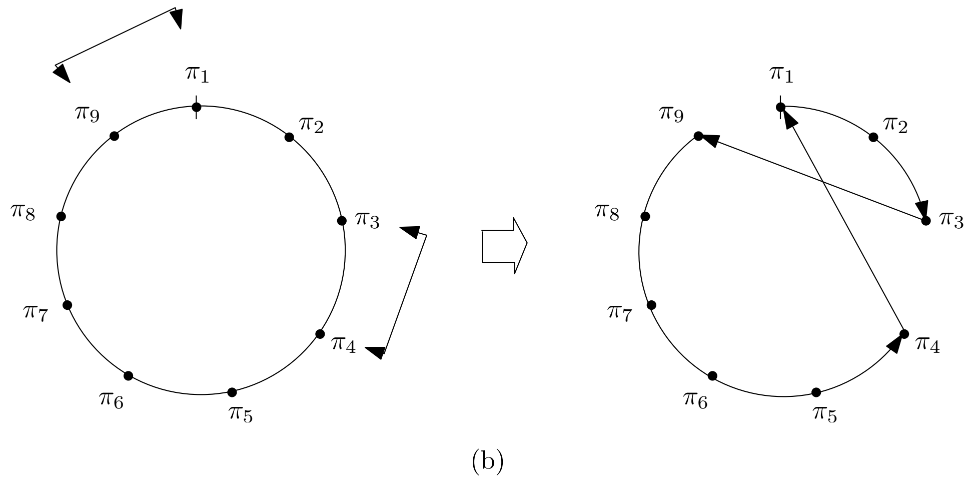


Figure 6.17: `2-opt move` permuting strategy.

6.5 Concluding remarks

We have given the definition of FSSs, shown how to construct a planar split network to represent a FSS, and that FSSs are a generalisation of affine split systems. We have explained how to use wiring diagrams as a tool to depict FSSs. The wiring diagram presentation also gives rise to several operations that can be used to search through the space of all FSSs. In the next chapter we will make use of these operations to help find weighted FSSs to represent a given distance matrix.

Chapter 7

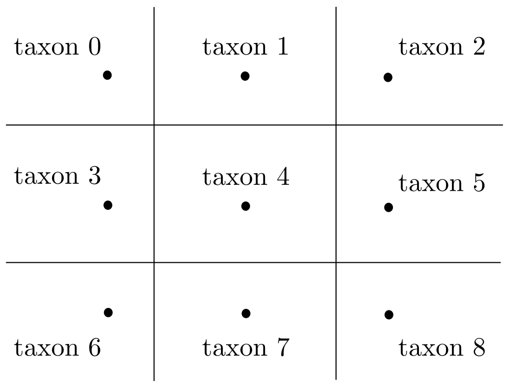
Representing distance matrices by planar split networks

7.1 Summary

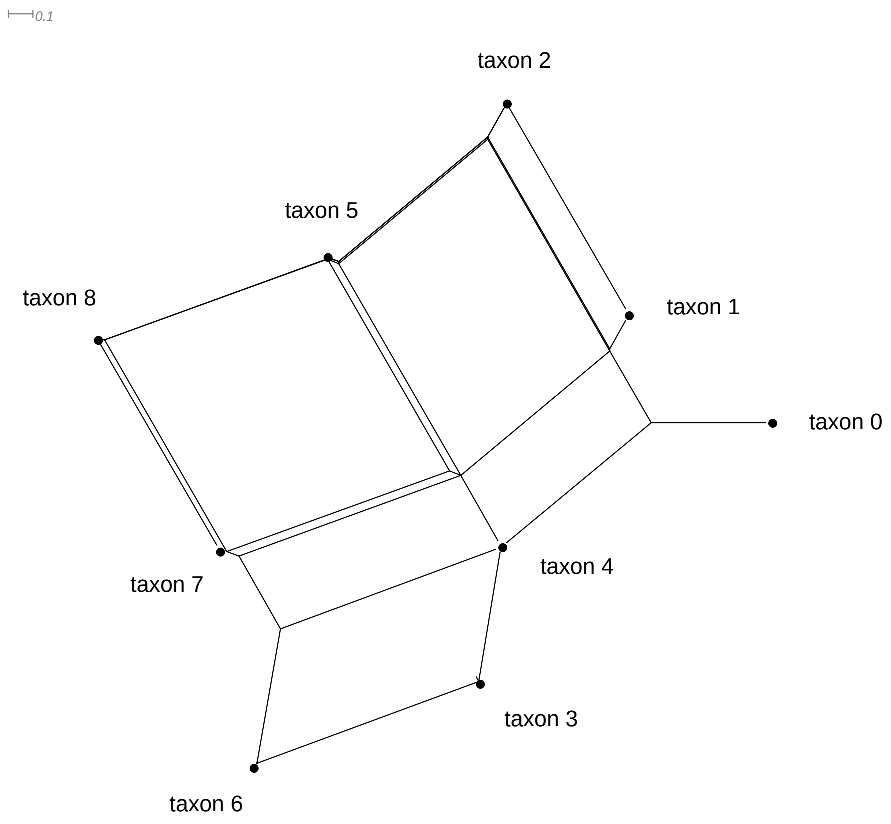
In Chapter 2 we have briefly discussed how NeighborNet can be used to obtain a planar split network from a distance matrix. However, there are cases for which the design of NeighborNet can cause problems. For example, consider the split system presented in Figure 7.1(a). If we input the associated distance matrix (not shown) into NeighborNet then we get the split network presented in Figure 7.1(b).

It can be seen that the split system produced by NeighborNet is not as simple as the original split system: There are 14 splits in the NeighborNet split system versus only 4 in the input. Also, the weights of the NeighborNet splits vary greatly in contrast to the constant weight of 1 over all 4 original splits. Moreover, although the original split system seems to suggest a symmetrical representation, there is no way to draw the NeighborNet network in such a balanced way.

It is not hard to see why NeighborNet has probably not done better in this example: The algorithm essentially tries to find a circular order of the taxa, then pushes them out to the perimeter of the network. However, in the example, taxon 4 lies right "inside" the other taxa, making it difficult for NeighborNet to cope with this situation.



(a)



(b)

Figure 7.1: (a) An affine split system; lines represent splits of the taxa, all splits have weight 1. (b) The corresponding split network produced by NeighborNet (using SplitsTree (84)).

In Chapter 6 we have introduced FSSs as a generalisation of affine split systems that can still be drawn on the plane. In this chapter we will make the first attempt at trying to derive a weighted FSS (and the corresponding planar split network) to represent a given distance matrix in the plane with the view to improving on NeighborNet for examples like the one in Figure 7.1. We will employ a local search approach which we will test on both random and "real life" data.

7.2 The MRF problem

A commonly used measure to quantify the difference between two distance matrices is *least squares*: Given matrices A and B of the same size $m \times n$, the least square difference between A and B, denoted as $||A, B||_2$, is defined as

$$||A, B||_2 = \sqrt{\sum_{\substack{i \in \{1,\dots,m\} \\ j \in \{1,\dots,n\}}} (a_{ij} - b_{ij})^2},$$

where a_{ij} and b_{ij} are the elements in row i and column j of matrices A and B, respectively. In this section we consider the following problem:

MatrixRepresentationWithFlatSplitSystems (MRF)

Input: a set X with n elements
a distance matrix d on X

Output: a full FSS $\mathfrak{S} = \{S_1, S_2, \dots, S_k\}$ on $X, k = \binom{n}{2}$
a weight vector $\omega = (\omega_1, \dots, \omega_k)$ of $\mathfrak{S}$ so that $||d_{\mathfrak{S},w}, d||_2$ is minimised.

This problem is motivated by the fact that NeighborNet and its "predecessor" Neighbor-Joining are known to optimise the Balanced Minimum Evolution (BME) length, i.e. they minimise the sum of the weights of all splits in the resulting split system (36; 96).

Note that since $||d_{\mathfrak{S},w}, d||_2 \geq 0$, if there is a split system $\mathfrak{S}$ and corresponding weight vector ω such that $||d_{\mathfrak{S},w}, d||_2 = 0$ then it is immediately clear that the pair $(\mathfrak{S}, \omega)$ is a best solution for MRF. Also note that requiring a full FSS as output is reasonable because if there is a non-full weighted FSS that minimises the least square difference between d and $d_{\mathfrak{S},\omega}$, as discussed in Chapter 6, one can always add more splits to make it full and assign weight 0 to the additional splits.

7.2.1 Computing the weights for a FSS to represent a distance matrix

To solve problem MRF, we propose searching through the "space" of FSSs using the operations introduced in the previous chapter. For each FSS we will compute the weights of the splits so that the least square difference is as small as possible, until, finally, a weighted FSS with globally minimum least square difference is be found. To break this process into pieces, we first consider the subproblem of fitting a distance matrix onto a given FSS:

WeightsComputingForFlatSplitSystems (WCF)

Input: a set X with n elements

a distance matrix d on X

a full FSS $\mathfrak{S} = \{S_1, S_2, \dots, S_k\}$ on $X, k = \binom{n}{2}$

Output: a weight vector $\omega = (\omega_1, \dots, \omega_k)$ of $\mathfrak{S}$ so that $||d_{\mathfrak{S},w}, d||_2$ is minimised

Following the approach used in NeighborNet, to solve this problem we construct a matrix T of size $k \times k$ where the rows correspond to the pairs of elements of X and the columns correspond to the splits of $\mathfrak{S}$ as follows:

$$T(\{x,y\}, S_i) = \begin{cases} 1 \text{ if } x, y \text{ are separated by } S_i, \\ 0 \text{ otherwise,} \end{cases}$$

for all $x, y \in X, i \in \{1, \dots, k\}$. Each column of matrix T is equivalent to a *split metric* defined in (22). The matrix is illustrated as in Table 7.1.

Now if we multiply T by ω, we have $[T \times \omega]$ being a k-element vector such that for any pair $\{x, y\}$ the entry $[T \times \omega](\{x, y\})$ gives the distance between x and y for all $x, y \in X$, i.e. $[T \times w] = d_{\mathfrak{S},\omega}$. Therefore, weight vector ω is a best solution, if there is any, of the following equation:

$$T \times \omega = d \tag{7.1}$$

	S_1	S_2	...	...	...	S_k
$\{x_1, x_2\}$	0/1	0/1	...	...	...	0/1
$\{x_1, x_3\}$	0/1	0/1	...	...	...	0/1
$\vdots$	...	...	...	...	...	...
$\{x_1, x_n\}$	...	...	...	...	...	...
$\{x_2, x_3\}$	...	...	...	...	...	...
$\vdots$	...	...	...	...	...	...
$\vdots$	...	...	...	...	...	...
$\vdots$	...	...	...	...	...	...
$\{x_{n-1}, x_n\}$	0/1	0/1	...	...	...	0/1

Table 7.1: An illustration for the matrix T.

Since T is a full rank matrix (22, Section 5.2), Equation (7.1) always has a unique solution

$$\omega = T^{-1}d, \tag{7.2}$$

where T^{-1} is the inverse matrix of T. We call such ω a *matching*, or *corresponding weight vector* of $\mathfrak{S}$. Note that Equation (7.2) makes it quite possible for ω to have negative elements whilst in general, one would want the weights of splits to be non-negative to have biological meaning. Depending on whether negative weights are accepted or not, we have different approaches to solve WCF.

If negative weights are acceptable, one can find the inverse matrix T^{-1} of T, then perform a matrix multiplication to get ω using Equation (7.2). However, in that case it is more efficient to solve Equation (7.1) using Gaussian elimination (67). We call this the ordinary least squares (OLS) approach.

If the non-negativity requirement is strict, we need to solve WCF with an additional constrain that $\omega \geq 0$. This extra condition makes WCF a non-negative least square problem. Lawson and Hanson (94, Chapter 23) analyse the problem intensively and propose an algorithm together with a FORTRAN implementation to solve it. We call this the constrained least square (CLS) approach.

7.2.2 Computing the error of a weighted FSS

Given a distance matrix d and a full FSS $\mathfrak{S}$, we have discussed how to derive a weight vector ω to get distance matrix $d_{\mathfrak{S},\omega}$. The following formula can be used to quantify how far $d_{\mathfrak{S},\omega}$ is from d, while avoiding the bias caused by the number of taxa:

$$\phi_{NNLS}(d, d_{\mathfrak{S},\omega}) := \frac{\sqrt{\sum_{\{x,y\}\in\binom{X}{2}} \left(d(x,y) - d_{\mathfrak{S},\omega}(x,y)\right)^2}}{\binom{n}{2}}. \tag{7.3}$$

We use this in case $d_{\mathfrak{S},\omega}$ is computed using the CLS approach. However, if the OLS approach is employed, for *every* FSS, by Equation (7.2), there is *always* a matching weight vector ω so that $\phi_{NNLS}(d, d_{\mathfrak{S},\omega}) = 0$. In other words, *any* full FSS is a best solution. For this reason we also use the following error measures:

1. Sum of all the weights, or ME (Minimum Evolution):
$$\phi_{ME}(d, d_{\mathfrak{S},\omega}) := \sum_{\{x,y\}\in X^2} \omega(\{x,y\}). \tag{7.4}$$

2. Sum of absolute values of all the weights, or SumAbs:
$$\phi_{SumAbs}(d, d_{\mathfrak{S},\omega}) := \sum_{\{x,y\}\in X^2} |\omega(\{x,y\})|. \tag{7.5}$$

3. Negative sum of positive weights, or NegSumPos:
$$\phi_{NegSumPos}(d, d_{\mathfrak{S},\omega}) := -\sum_{\substack{\{x,y\}\in X^2\\ \omega(\{x,y\})>0}} \omega(\{x,y\}). \tag{7.6}$$

The ME measure has been used to search through the space of phylogenetic trees, i.e. for compatible split systems (61; 89; 126). Therefore it is natural to use it for general split systems. SumAbs is a variation of ME that has been mentioned in (77). For NegSumPos, we actually want to maximise the influence of the positive weights by maximising their sum; with the definition of NegSumPos we keep this intention by minimising the measure. This makes our theme more consistent in that we try to minimise all error measures.

7.2.3 Local search strategy

We are now ready to present an heuristic local search solution to the MRF problem. Here we want the weights of the final splits to be non-negative for convenient biological interpretation.

1. Start with a weighted full FSS $\mathfrak{S}^0 = \mathfrak{S}(\mathcal{P})$ with corresponding p-sequence $\mathcal{P}$ and weight vector ω^0.

2. Let $\varepsilon = \phi(d, d_{\mathfrak{S}^0,\omega^0})$ where ϕ is one of the error measures defined in Section 7.2.2.

3. Use the operations presented in Chapter 6 to obtain the set of neighbors (or a neighborhood) $\mathcal{N}$ of $\mathcal{P}$.

4. For d and each $\mathcal{R} \in \mathcal{N}$ solve problem WCF using one of the two approaches presented in Section 7.2.1 to get FSS $\mathfrak{S}(\mathcal{R})$ and corresponding weight vector $\omega_{\mathfrak{S}(\mathcal{R})}$. Note that the approach must match the error measure decided in Step 2, as discussed above.

5. Let $\sigma = \min\{\phi(d, d_{\mathfrak{S}(\mathcal{R}),\omega_{\mathfrak{S}(\mathcal{R})}})|\mathcal{R} \in \mathcal{N}\}$.

6. If $\sigma < \varepsilon$ then let $\varepsilon = \sigma, \mathcal{P} = \arg\min\{\phi(d, d_{\mathfrak{S}(\mathcal{R}),\omega_{\mathfrak{S}(\mathcal{R})}})\}$ and go back to Step 3, otherwise go to Step 7.

7. If the OLS approach is employed, solve problem WCF with d and $\mathfrak{S}(\mathcal{P})$ as the input to compute the (non-negative) weights .

Note that if the OLS approach is employed, i.e. the error measure ε chosen at Step 2 is either ϕ_{ME}, ϕ_{SumAbs}, or $\phi_{NegSumPos}$, then Step 4 may yield negative weights in exchange for faster calculations. In that case Step 7 is performed to ensure the non-negativity of the weights.

The efficiency of the search depends on several factors: The choice of the initial split system, the use of the operations to generate the neighborhood at Step 3, and, in case of the OLS approach, the error measure to be used. We will discuss the first two factors in the beginning of the next section. Below we propose several strategies to combine the operations presented in Chapter 6 to generate a neighborhood of a given p-sequence.

- Flipping and Shifting (FliShift) : For every viable triple, use `Shifting` to shift to the position of the first element of the triple, then use `Flipping` to get a new p-sequence. The size of the neighborhood created by `FliShift` is $O(n^2)$ because operation `Flipping` can start at at most $k = \binom{n}{2}$ positions of the sequence of swapping points.

- Permuted Flipping (PermFlip) : Use `Flipping` on the sequence of swapping points to generate a pool of sequences of swapping points. Use `Permuting` on the initial permutation to generate a pool of permutations. Match each new sequence of swapping points with a new permutation to generate a new p-sequence. Note that the neighborhood generated by this strategy is generally big: Even if the simplest `Permuting` strategy, i.e. `Exchange`, is used, the size of the neighborhood is $O(n^3)$ already.

- FliShift and Permuting (PermFliShift) : The neighborhood created by this strategy is the union of two sets of p-sequences. The first set contains those generated by strategy FliShift. The second set is obtained by using Permuting on the initial permutation to generate a pool of permutations, then matching each of such permutations with the original sequence of swapping points. It can be observed that the PermFliShift neighborhood is bigger than that of FliShift but is still much smaller than that of PermFlip. For example if `Pair-Exchange` is employed as the permuting method, then the neighborhood is still of size $O(n^2)$ comparing to $O(n^4)$ of PermFlip.

One can check that all three neighborhood-generating strategies above are able to explore large portions of the space of FSSs. Moreover, referring to Theorem 13, it is evident that FliShift can explore all sequences of swapping points and PermFliShift can explore the whole FSS space. In the next section we will describe the implementation of the local search with all these strategies.

7.3 An initial evaluation of the new approach

We implemented the local search framework described in Section 7.2.3 for both the CLS and OLS approaches using Java. The code for Lawson and Hanson's NNLS algorithm is adapted from Spillner's work, which in turn is an adaptation from Bryant's work (private communication). The code to do Gaussian elimination is the work of Raphael (119).

Theoretically the CLS approach can always be used but the NNLS algorithm takes a lot of running time whilst the Gaussian elimination is much faster. Therefore in these tests we basically use the CLS approach to get a feeling for how much improvement can be made, and the OLS approach to see how the approach might work in practice.

In the implementation, for the initial FSS, we use the full circular split system constructed by NeighborNet. In particular, we input distance matrix d into NeighborNet and get permutation π_{NN} corresponding to the circular order of the taxa. Let $P = (1, 2, \ldots, n-1, 1, 2, \ldots, n-2, \ldots, 1, 2, 1)$. Then $\mathcal{P} = (\pi_{NN}, P)$ is a p-sequence representing the input FSS. The NeighborNet split system is chosen as the input because we would like to compare our approach to this popular method. There are other ways to choose the starting split system but they are for future work.

Of the permuting methods that could be used to generate the neighborhood during the search, based on the analysis of the sizes of the neighborhood in Section 7.2.3, we chose to use only `Exchange` for PermFlip in CLS approach, the three strategies `Exchange, Pair-Exchange, 2-opt move` for PermFlip in OLS approach, and PermFliShift in both CLS and OLS approaches. For the OLS approach, we tried all three error measures ϕ_{ME}, ϕ_{SumAbs}, and $\phi_{NegSumPos}$.

All in all, there were 26 parameter sets to run the search with for each data set: 5 with the CLS and 21 with the OLS approaches. For brevity, we represent each choice of parameters as a tuple, e.g. (CLS, PermFlip, `Exchange`) corresponds to the CLS approach with PermFlip as the neighborhood generating strategy and `Exchange` as the permuting method, and (OLS, PermFliShift, ϕ_{ME}, `2-opt move`) corresponds to the OLS approach with PermFliShift as the neighborhood generating strategy, ϕ_{ME} as the error measure (refer to Step 4 in the strategy represented in Section 7.2.3), and `2-opt move` as the permuting method.

In this framework, for the example shown in Section 7.1, the local search finds the original split system using (CLS, PermFlip, `Exchange`). The planar network representing the resulting split system is represented in Figure 7.2 (this network

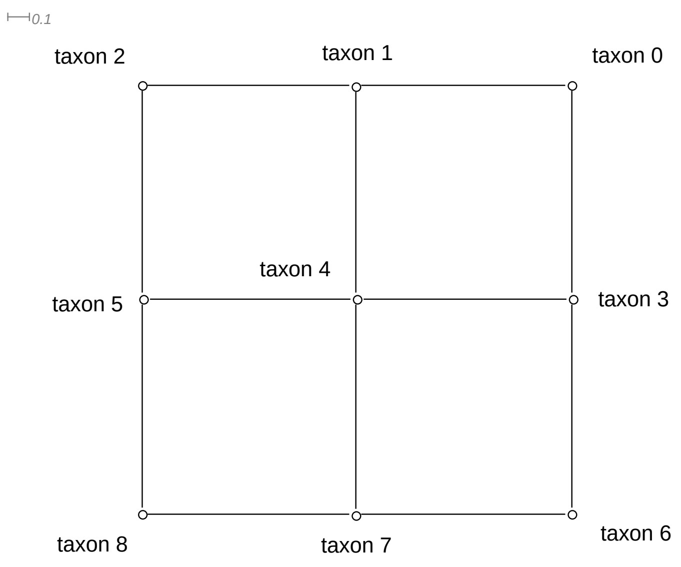


Figure 7.2: The planar split network representing the best FSS found by the local search for the distance matrix derived from the split system represented in Figure 7.1(a).

and all other planar networks to represent FSSs in this thesis are drawn using a program developed by Spillner (private communication)). The NNLS error is $\phi_{NNLS} \approx 8.36E - 7$ compared to 0.05 for the NeighborNet split system.

To further evaluate the new approach we made some systematic tests on random data as well as trying it out two "real life" examples. For the random data, we consider two sub-types: Tests on randomly generated distance matrices, which we present next, and tests on so-called random *flat distance matrices*, i.e., matrices induced from randomly generated FSSs, which we present in Section 7.3.2. We present the real life examples in Sections 7.3.3 and 7.3.4.

7.3.1 Random distance matrices

We group the tests by the number of taxa. We will consider "small" data sets of 5, 6, 7, 8, 9, 10 taxa and "bigger" data sets of 15, 20, 25 and 30 taxa. For each number of taxa, we generate 20 random distance matrices using the built-in pseudo random number generator in Java. These matrices are symmetrical with values uniformly distributed between 0 and 1.

After all the tests are carried out, the final results are tabulated. Let m be the number of data sets with a given number of taxa. Let $\mathfrak{S}_1$ be the full circular split system induced by NeighborNet when fed with distance matrix d, ω_1 be the weight vector found when solving WCF with d and $\mathfrak{S}_1$ as the input. Let $\mathfrak{S}_2, \omega_2$ be the weighted FSS found by the local search. We now define the terms that appear in the columns heading the resulting tables:

- err_{NN}: The average NNLS error of $d_{\mathfrak{S}_1,\omega_1}$:

$$err_{NN} = \frac{\sum \phi_{NNLS}(d, d_{\mathfrak{S}_1,\omega_1})}{m}$$

- err_{FSS}: The average NNLS error of $d_{\mathfrak{S}_2,\omega_2}$:

$$err_{FSS} = \frac{\sum \phi_{NNLS}(d, d_{\mathfrak{S}_2,\omega_2})}{m}$$

- FSS/NN: The average percentage between the errors of $\mathfrak{S}_2$ and $\mathfrak{S}_1$. This measure quantifies the significance of the improvement made by the FSS: The smaller FSS/NN the more significant the improvement.

$$FSS/NN = 100 \times \frac{\sum \Big(\phi_{NNLS}(d, d_{\mathfrak{S}_2,\omega_2}) / \phi_{NNLS}(d, d_{\mathfrak{S}_1,\omega_1}) \Big)}{m}$$

- Steps: The average number of neighborhoods explored during the search.

- FSSs: The average number of *unique* FSSs considered during the search. During the search there can be different p-sequences representing the same FSS generated, we only count one of them.

- S_s: The average percentage of splits shared between $\mathfrak{S}_1$ and $\mathfrak{S}_2$:

$$S_s = 100 \times \frac{\sum \Big(|\mathfrak{S}_1 \cap \mathfrak{S}_2| / |\mathfrak{S}_1 \cup \mathfrak{S}_2| \Big)}{m}$$

- W_s: The average percentage of the total weight of splits shared between $\mathfrak{S}_1$ and $\mathfrak{S}_2$:

$$W_s = 100 \times \frac{\sum \frac{\sum_{S \in \mathfrak{S}_1 \cap \mathfrak{S}_2} (w_1(S) + w_2(S))}{\sum_{S \in \mathfrak{S}_1} w_1(S) + \sum_{S \in \mathfrak{S}_2} w_2(S)}}{m}$$

Note that in the last formula, the bigger $\sum$ indicates the sum over the m data sets, the smaller $\sum$ represents the sum over all splits in each of the split systems corresponding to those data sets.

If the OLS approach is employed, we add column $Wins$ to show the percentage of times where the local search does find a FSS that has the NNLS error smaller than or equal to that of the full circular split system constructed by NeighborNet. In this case, all quantities defined above are computed only for the data sets on which the improvements are made. We need the additional column $Wins$ because when using the OLS approach, we actually optimise some different measures hoping that the output FSS also has a good NNLS error. However, since the alternative measures are not necessarily 100% correlated to the NNLS error, the FSS with the best alternative error, after having its split weights recalculated in Step 7 for non-negativity, may still have a worse NNLS error than that of the NeighborNet split system.

We now present the results. For each of the 26 sets of parameters for the local search, we obtain a resulting table but our discussion only concentrates on the results for the choices of parameters which gave the highest average FSS/NN values. In particular, for CLS approach, the best results, shown in Table 7.2, are achieved with (CLS, PermFliShift, `2-opt move`).

It can be observed that the FSSs make some improvements over the NeighborNet split systems but these are somewhat limited: The FSS errors are between 93.5-98% (with an average of 95.84%) of the NeighborNet error. As we can see in columns “Steps” and “FSSs” of Table 7.2, the average number of neighborhoods explored during the search process increases from 2 to nearly 10 with the size of the data increasing, and consequently, the number of FSSs explored increases fairly quickly. This is an indication that the search does work in a sensible way.

[1]We can get the results on only 9 data sets of 30 taxa due to running time.

n	err_{NN}	err_{FSS}	FSS/NN	Steps	FSSs	S_s	W_s
5	0.030594	0.029700	94.89	2.00	19.80	89.52	98.42
6	0.027754	0.026295	95.49	2.20	32.60	81.38	93.33
7	0.028372	0.027279	96.51	2.35	49.00	81.37	93.40
8	0.027216	0.025726	95.03	3.10	85.00	67.16	88.32
9	0.028582	0.026659	93.44	3.45	122.15	60.52	82.70
10	0.025267	0.023977	94.91	3.30	146.85	59.96	84.20
15	0.020117	0.019318	96.05	5.65	587.40	42.64	70.58
20	0.016071	0.015581	96.95	7.00	1321.80	32.79	63.55
25	0.013705	0.013300	97.05	9.75	2910.45	27.45	55.73
30[1]	0.011633	0.011413	98.11	9.67	4191.89	29.40	59.45

Table 7.2: Results on randomly generated distance matrices with (CLS, PermFliShift, `2-opt move`).

Nevertheless, it should be noticed that the number of unique FSSs explored by the search accounts for only a very small portion over all possible FSSs. Though we haven't been able to count the exact number of all unique FSSs, it has been known (129; 140) that the number of all binary phylogenetic trees on a set of n taxa is

$$b(n) = (2n-5)!! = 1 \times \times 5 \times \cdots \times 2n-5) = \frac{(2n-4)!}{(n-2)!2^{(n-2)}}.$$

For example if $n = 30$, we have $b(30) \approx 9 \times 10^{36} >> 4200$. Also note that the number of FSSs must be greater than that of the binary phylogenetic trees. Therefore it is quite possible that the search got trapped in a local optimum: Since we always start with the NeighborNet-produced split system, which might be near a local optimum, the search could just have just got a bit better and then got stuck at the local optimum without being able to find the global one which lies in another region (see Figure 7.3). There are several strategies to cope with this problem but we have been unable to investigate these further in this thesis due to time constraints.

Something interesting can be concluded from columns "S_s" and "W_s": We find that the average percentage of splits shared between the original and the resulting FSSs decreases fairly fast, and so does the contribution of the weights of those

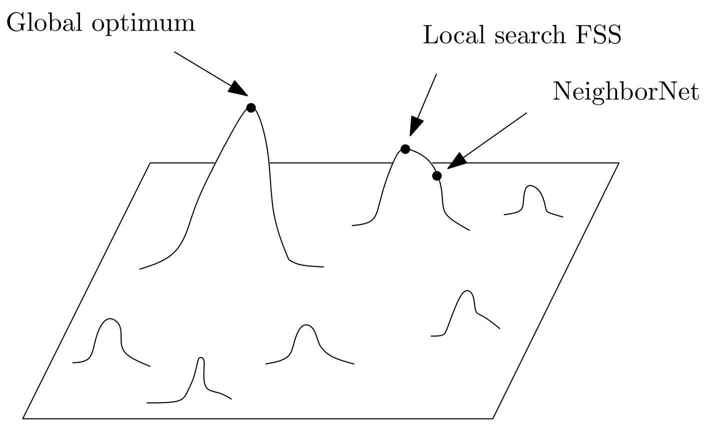


Figure 7.3: The local search might get stuck at a local optimum near to the position found by NeighborNet.

shared splits, though not quite as fast. This means that while both pretty well accommodate the input distance matrices, the FSSs can be quite different from the split systems found by NeighborNet. In other words there may be many FSSs with good fitness given a distance matrix, and the one produced by NeighborNet is only one of them. The lower decreasing rate of the percentage of total weights of the shared splits is also noticeable. This means that the NeighborNet split systems and the corresponding FSSs share splits of higher weights, which are probably the more important splits. This could suggest that given a distance matrix there may be a set of "core" splits that would appear in any "good" split system.

When applying the OLS approach, we obtain more diverse results. In particular, on small data sets (no more than 10 taxa), best results are obtained with (OLS, FliShift, ϕ_{SumAbs}), see Table 7.3, whilst on bigger data sets (from 15 to 30 taxa), they are obtained with (OLS, PermFliShift, ϕ_{ME}, `Exchange`), see Table 7.4.

A closer examination reveals some interesting facts. First, the set of parameters that work well on the smaller data sets do not work well on bigger data sets and vice versa. More specifically, in Table 7.3, improvements are made on 30% to 70% of the small data sets and average 50% over all these data. However, on the 15-taxa data, only 15% of the time the search finds a FSS that has the same error as the NeighborNet split system. It is even worse for data sets

n	Wins	err_{NN}	err_{FSS}	FSS/NN	Steps	FSSs	S_s	W_s
5	50.00	0.023264	0.022132	92.00	1.88	10.00	92.22	98.62
6	70.00	0.030830	0.030199	98.36	1.85	11.85	89.75	94.66
7	55.00	0.028209	0.027933	99.20	2.25	16.12	92.42	97.53
8	55.00	0.028532	0.027897	98.19	2.71	22.29	84.55	93.82
9	30.00	0.031145	0.030501	98.13	2.33	21.67	81.63	91.60
10	40.00	0.025444	0.025170	98.87	2.33	23.83	88.95	95.08
15	15.00	0.018594	0.018594	100.00	2.00	31.00	97.71	99.95
20	0.00	-	-	-	-	-	-	-
25	0.00	-	-	-	-	-	-	-
30	5.00	0.012005	0.012004	100.00	2.00	61.00	92.98	99.64

Table 7.3: Best results on small random distance matrices obtained with (OLS, FliShift, ϕ_{SumAbs}).

of 20 taxa and above: (OLS, FliShift, ϕ_{SumAbs}) almost always finds FSSs that have bigger errors than those constructed by NeighborNet. In contrast, as shown in Table 7.4, (OLS, PermFliShift, ϕ_{ME}, `Exchange`) works very badly on small data sets, making improvements between only 0% and 15% (average 7.5%), but it performs quite well on big data sets with improvements between 20% and 50% (average 37.5%).

Nevertheless we should probably not expect too much improvement on the error in view of the humble error reduction made with the CLS approach. In fact, on big data sets, the FSS errors are just under 99% to just under 100% of the NeighborNet errors, and on small data sets, with the exception of those with 5 taxa, the FSS errors are between 94.5% and 99.2% of the NeighborNet errors. These very limited improvements on the error are probably not surprising because of the following reasons: (1) NeighborNet does well already as seen in Table 7.2, (2) The search starts with the NeighborNet split system, and (3) In OLS-based aproach, we use an alternative measure during the search in order to optimise the NNLS measure. Though reasonable, these measures might not be the best alternatives. In particular, the measure SumAbs is practically inapplicable: Of the 4 sets of parameters involving SumAbs, none helps find a FSS that has a smaller error than NeighborNet.

n	Wins	err_{NN}	err_{FSS}	FSS/NN	Steps	FSSs	S_s	W_s
5	0.00	-	-	-	-	-	-	-
6	5.00	0.041778	0.039882	95.46	2.00	23.00	27.27	42.10
7	5.00	0.026446	0.023675	89.52	2.00	29.00	56.25	67.54
8	15.00	0.030857	0.029082	94.91	2.33	36.00	72.46	88.88
9	15.00	0.032997	0.032331	98.15	2.00	37.00	66.80	86.78
10	5.00	0.023709	0.021969	92.66	3.00	58.00	47.83	68.36
15	50.00	0.020205	0.019990	98.93	3.11	91.78	69.11	86.36
20	35.00	0.016411	0.016367	99.73	3.00	119.43	78.47	92.66
25	20.00	0.014007	0.013978	99.80	3.67	182.00	71.37	89.50
30	45.00	0.011531	0.011502	99.76	4.75	281.62	66.33	88.59

Table 7.4: Best results on small random distance matrices obtained with (OLS, PermFliShift, ϕ_{ME}, `Exchange`).

Table 7.4 reveals another interesting fact: The percentages of splits shared between the NeighborNet split systems and the resulting FSSs does *not* really decrease when the size of the data sets increases. What this might suggest is that the local search with this specific set of parameters works more stably, independently on the data size. With an average of 61.77% over all tested matrices, the percentages of shared splits are small enough to support the previous hypothesis that there may be many split systems that well accommodate an input distance matrix. The last columns also still supports the idea of "core" splits.

When comparing Table 7.3 and 7.4 with Table 7.2, it can be seen that the numbers of neighborhoods and FSSs explored during the OLS-based search are smaller than those during the CLS-based search. One may argue that if more neighborhoods, and, hence, more FSSs were considered, then the percentage of cases with improvements would be higher. This is generally true but in our current search framework, it is not really necessarily suitable. In Table 7.5 we present the second best results which are achieved with (OLS, PermFlip, ϕ_{SumAbs}, `Exchange`). It can be seen that more neighborhoods, and consequently more FSSs, are explored but on small data sets, even though quite reasonable, the results are not as good as those in Table 7.3, and on bigger data sets, they are much worse than those in Table 7.4. Therefore, guiding the search to look into

n	Wins	err_{NN}	err_{FSS}	FSS/NN	Steps	FSSs	S_s	W_s
5	70.00	0.026445	0.025430	94.15	2.25	20.88	83.62	96.78
6	65.00	0.027732	0.026216	95.69	2.27	25.91	74.41	89.58
7	30.00	0.023858	0.022578	95.88	3.20	41.60	81.21	90.22
8	30.00	0.024805	0.024124	97.43	2.80	43.20	63.71	82.56
9	15.00	0.027125	0.025221	92.88	3.75	63.25	48.17	77.91
10	25.00	0.026216	0.025560	97.35	2.50	50.00	48.82	73.93
15	5.00	0.023430	0.023178	98.92	6.00	172.00	43.59	76.09
20	5.00	0.018096	0.018027	99.62	5.00	195.00	45.45	76.30
25	5.00	0.013581	0.013553	99.79	14.00	39298.00	40.58	76.05
30	5.00	0.011795	0.011794	99.99	10.00	42610.00	28.12	47.18

Table 7.5: The second best results on random distance matrices obtained with (OLS, PermFlip, ϕ_{SumAbs}, `Exchange`).

a more reasonable neighborhood is probably better than going to a big but not much related one.

7.3.2 Random flat distance matrices

We perform a similar process to that in Section 7.3.1 but the input distance matrices are induced from randomly FSSs. We did this as we hoped the local search strategy would work better for input data that is more similar to the output that we are generating. The random FSSs are generated with corresponding random initial permutations and random sequences of swapping points as follows: The random permutation is initialised as the identity permutation $(1, 2, \dots n)$. Repeating n times, at time i swap element i with a random element lying between $i{+}1$ and n. The elements of the random sequence of swapping points are generated one by one from 1 to k where $k = \binom{n}{2}$. For element $j, 1 \leq j \leq k$, on permutation $\Pi = (\pi_1, \pi_2, \dots, \pi_n)$, starting as the initial permutation, we identify a random position i such that π_i and π_{i+1} have not been swapped with each others, then we assign value i to swapping point j and adjust permutation Π by swapping π_i and π_{i+1}. One can check that these always produce a valid pair of initial permutation and sequence of swapping points.

Since the input distance matrix comes from a FSS, we can ideally expect the local search to find a FSS that perfectly fit the matrix. For this reason we will introduce some more measures to help evaluate the results. Let $\mathfrak{S}_0, \omega_0$ be the random weighted FSS from which distance matrix d is induced, we define the following additional values:

- S_{rNN}: The average percentage of splits in $\mathfrak{S}_0$ that are recovered in $\mathfrak{S}_1$:

$$S_{rNN} = 100 \times \frac{\sum \Big(|\mathfrak{S}_0 \cap \mathfrak{S}_1| / |\mathfrak{S}_0| \Big)}{m}$$

- W_{rNN}: The average contribution to $\mathfrak{S}_0$ of the splits recovered by $\mathfrak{S}_1$:

$$W_{rNN} = 100 \times \frac{\sum \frac{\sum_{S \in \mathfrak{S}_0 \cap \mathfrak{S}_1} w_0(S)}{\sum_{S \in \mathfrak{S}_0} w_0(S)}}{m}$$

- S_{rFSS}: The average percentage of splits in $\mathfrak{S}_0$ that are recovered in $\mathfrak{S}_2$:

$$S_{rFSS} = 100 \times \frac{\sum \Big(|\mathfrak{S}_0 \cap \mathfrak{S}_2| / |\mathfrak{S}_0| \Big)}{m}$$

- W_{rFSS}: The average contribution to $\mathfrak{S}_0$ of the splits recovered by $\mathfrak{S}_2$:

$$W_{rFSS} = 100 \times \frac{\sum \frac{\sum_{S \in \mathfrak{S}_0 \cap \mathfrak{S}_2} w_0(S)}{\sum_{S \in \mathfrak{S}_0} w_0(S)}}{m}$$

Again, in the formulas for W_{rNN} and W_{rFSS}, the bigger $\sum$ indicates the sum over the m data sets, the smaller $\sum$ represents the sum over all splits in each of the involved split systems.

As with the test on random distance matrices, for each approach, we only present the results where the improvements are best in term of the NN/FSS ratio. For the CLS-based approach, the best results, shown in Table 7.6, are obtained with (CLS, PermFliShift, `2-opt move`). The first and most important observation is that the improvement on errors is quite reasonable: The FSS errors are mostly between 65-75%, go down to just 7.65% on 5-taxa, and average to only 57.4% of the NeighborNet error. This fact suggests two important points: (1) NeighborNet does not work as well on flat distance matrices as it does on general ones, and (2) FSSs can be used to make significant improvements over NeighborNet on flat distance matrices.

Table 7.6 also indicates that both the number of neighborhoods and the number of FSSs explored increases with the size of the tests. Thus it seems that the search does survey a good range of the FSS space. However, these quantities alone cannot prevent the possibility of the search getting stuck in a local optimum as analysed in Section 7.3.1.

Columns "S_s" and "W_s" agree with the two hypotheses drawn in the tests with random distance matrices, namely that there may be many split systems that well represent one distance matrix, and there may be a group of "core" splits given a matrix. However, comparing columns S_s in Table 7.6 and 7.2 one can observe that on flat distance matrices, the FSSs and NeighborNet split systems share more "core" splits than on random distance matrices.

Since we start with FSSs, it is natural to expect that the search recovers many original splits. However, columns "S_{rNN}" and "S_{rFSS}" show that the percentages of splits recovered by both NeighborNet and the local search on the small data sets are reasonably high but they decrease fairly quickly on bigger data sets. At first glance the reader may find this somewhat disappointing. However, a closer examination reveals a possible explanation: All the input FSSs are generated in full, i.e. the number of input splits increases in quadratic, whilst the number of splits with non-zero weights found both NeighborNet and local search increases much more slowly (data not shown). The finding suggests that both NeighborNet and the local search tend to find split systems as compact as possible while trying to maintain the optimization criteria. We think this feature is promising as we have never seen any real taxa set where all splits possible are needed. Therefore a similar test where only a portion of the generated FSSs has non-zero weights will be useful in the future work.

Columns "W_{rNN}" and "W_{rFSS}" when compared with columns "S_{rNN}" and "S_{rFSS}" respectively indicate that both NeighborNet and our search do recover important splits of the original system. For example on the 25-taxa data sets, on average NeighborNet recovers 21.61% of the splits but those splits contribute to 54.82% the total weight of the input split system; the corresponding figures with our local search are 25.5% and 59.6%. It can also be seen that our search recovers about 4% more splits than NeighborNet does. These findings give further support to the hypotheses about the group of "core" splits proposed before. If more time were available, it would be interesting to do a further comparison between three split systems: The original flat one, the one produced by NeighborNet and the one found by our local search.

n	err_{NN}	err_{FSS}	FSS/NN	Steps	FSSs	S_s	W_s	S_{rNN}	W_{rNN}	S_{rFSS}	W_{rFSS}
5	0.003951	0.000871	7.65	1.25	13.35	92.27	96.84	87.01	94.70	92.73	97.42
6	0.025709	0.004917	12.81	1.90	28.90	80.44	92.24	76.51	90.70	83.08	94.93
7	0.047231	0.028209	48.29	1.95	41.35	80.26	92.06	69.77	86.87	75.01	90.47
8	0.065845	0.047355	67.94	1.95	54.70	79.96	91.22	63.73	82.43	67.82	85.72
9	0.087002	0.061405	69.13	2.30	82.95	75.76	88.50	58.55	80.48	60.03	82.41
10	0.100296	0.071962	74.20	2.45	110.10	75.57	88.98	53.25	78.21	55.42	80.15
15	0.202519	0.153722	77.85	4.00	416.80	68.32	84.61	36.36	67.23	40.89	69.91
20	0.316351	0.231736	76.29	5.15	973.15	59.86	79.95	27.04	59.62	31.18	63.65
25	0.406801	0.293755	73.99	7.65	2285.20	49.88	72.80	21.61	54.82	25.42	59.68
30[1]	0.604689	0.378319	65.75	8.31	3603.23	38.71	64.23	16.01	49.78	20.00	55.26

Table 7.6: The best results on "flat" distance matrices when using the CLS approach obtained with (CLS, PermFliShift, `2-opt move`).

[1]We can get the results on only 14 data sets of 30 taxa due to running time.

For the OLS approach, we found two sets of parameters that give interesting results. First, in terms of the number of times improvements are made, the best choice of parameters is (OLS, PermFlip, ϕ_{ME}, `Exchange`) (Table 7.7). The search makes improvements from 65% to 100% (average 89.5%). On data sets containing up to 15 taxa the corresponding numbers are even better: 90%, 100% and 95.7%, i.e. improvements are obtained most of the times. On data sets of 20, 25, and 30 taxa, the local search outperforms NeighborNet in less cases but it still does quite well compared to working on random distance matrices, with improvements made on 75%, 85% and 65% of the case, respectively.

On the significance of the improvements, the average NNLS errors of the FSSs by test sizes are from 55% to 98% that of the NeighborNet split systems with the exception of only 6.27% on 5-taxa data sets. The average FSS/NN measure over all data sets is 73.85% and is very reasonable: It means on that average, on all tested data, the NNLS error of the FSSs found by local search is under 3/4 that of the NeighborNet split systems.

However, it must be noted that the significance of the improvement decreases when the test size increases, and there seems to be a clear cut between the smaller and bigger data sets. The average FSS/NN ratio on the smaller data is 60.13% while that on the bigger data is 94.43%. These numbers suggest that the current parameter set may not be good enough to use on large data sets an thus is less applicable. It may also be necessary to find better ways than NeighborNet for choosing an initial split system.

The second set of parameters we would like to discuss is (OLS, PermFliShift, ϕ_{ME}, `2-opt move`), see Table 7.8. This does better than or equal to NeighborNet in 55% to 100% (average 85%). Although 85% is slightly less than 89.5% attained with (OLS, PermFlip, ϕ_{ME}, `Exchange`), this parameter set works more stably when the data size increases. For example, on data sets of 30 taxa, the local search with (OLS, PermFliShift, ϕ_{ME}, `2-opt move`) makes an improvement in 90% of the cases compared to just 65% obtained with the parameter set discussed above. It does not make improvement on as many cases on 5-taxon data sets (55% vs 95%) but we would still prefer it for the ability to perform better on bigger data sets.

More importantly, the reduction of errors is more significant than that achieved with the other parameter set for the OLS approach, especially on bigger data sets. The NNLS errors of the FSSs found by the local search with (OLS, PermFliShift, ϕ_{ME}, `2-opt move`) range from 9.11% to 87% (average 64.78%) those of the NeighborNet split systems. The average FSS/NN measures of 64.78% over all data sets and 85.94% on bigger data sets are significantly better than corresponding figures of 73.85% and 94.43% obtained with (OLS, PermFlip, ϕ_{ME}, `Exchange`). Even though not as good as the results produced with the CLS approach, whose corresponding numbers are 57.4% and 73.47%, (OLS, PermFliShift, ϕ_{ME}, `2-opt move`) makes a good compromise between the quality of the results and the running time.

In Figure 7.4, we present a visual illustration for the significance analysis: Compared to (OLS, PermFlip, ϕ_{ME}, `Exchange`), (OLS, PermFliShift, ϕ_{ME}, `2-opt move`) does worse on 5 taxa, slightly worse on 10 taxa, but better and significantly better on all remaining data sizes. The latter performs slightly better than (CLS, PermFliShift, `2-opt move`) on two cases but worse on the remaining cases. In general, it does significantly better than (OLS, PermFlip, ϕ_{ME}, `Exchange`), and worse than (CLS, PermFliShift, `2-opt move`) whose cost is much higher.

Examining columns S_s and W_s in Tables 7.7 and 7.8, again we observe that the NeighborNet and our local search both find splits of high weights. However, the second parameter sets seem to find FSSs with less shared splits, i.e. it seems to find more “core” splits than the first parameter set. The last four columns in those tables also agree with what we proposed when analysing the results obtained with the CLS approach: There can be many split systems that well accommodate a distance matrix; both our local search and NeighborNet seem to be able to recover “core” splits of the original FSS, but our search recovers about 3-4% more such splits.

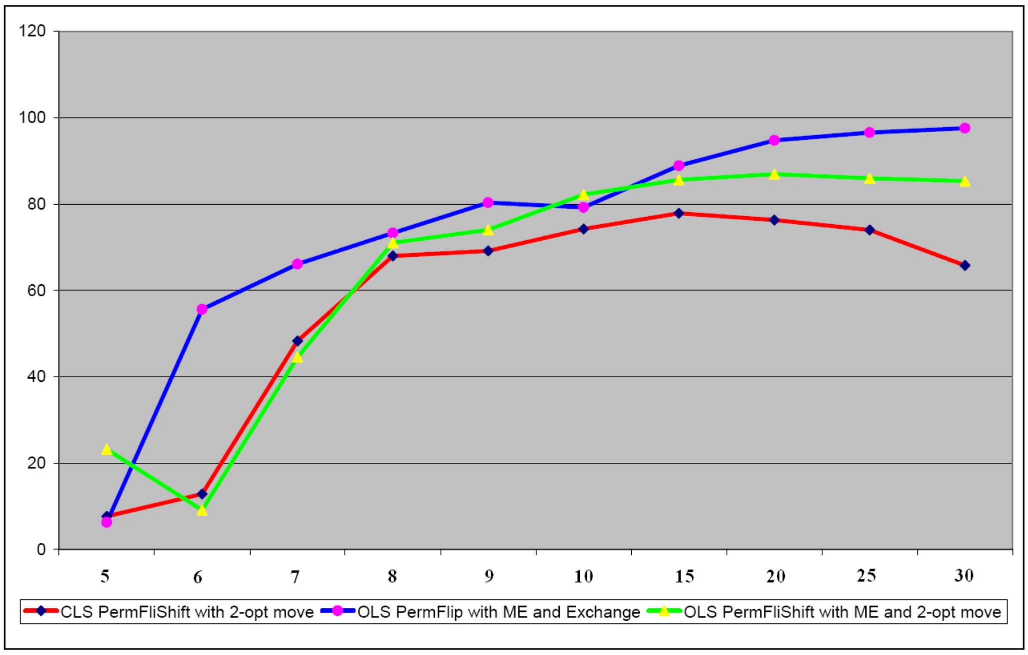


Figure 7.4: The average improvements on "flat" distance matrices obtained by the local search with different parameter sets. x-axis labelled by the data size, y-axis labelled by the FSS/NN measure.

n	Wins	err_{NN}	err_{FSS}	FSS/NN	Steps	FSSs	S_s	W_s	S_{rNN}	W_{rNN}	S_{rFSS}	W_{rFSS}
5	95.00	0.004159	0.001269	6.27	1.33	27.39	91.39	96.48	86.32	94.42	91.11	96.41
6	90.00	0.027189	0.015897	55.60	1.47	50.32	90.19	96.64	76.06	90.53	80.74	92.36
7	100.00	0.047231	0.030072	66.08	1.70	87.50	86.58	94.65	69.77	86.87	73.97	88.90
8	95.00	0.063896	0.046729	73.29	2.06	155.28	81.26	91.93	64.02	82.88	68.03	85.48
9	100.00	0.087002	0.065863	80.30	2.65	284.80	77.28	89.34	58.55	80.48	61.39	82.30
10	100.00	0.100296	0.076439	79.21	2.60	365.25	78.72	90.56	53.25	78.21	56.95	80.20
15	90.00	0.199523	0.178446	88.88	3.17	1225.33	73.64	89.42	37.35	67.87	41.41	69.81
20	75.00	0.323476	0.307912	94.73	4.36	3304.43	78.53	92.09	26.37	58.85	28.33	60.19
25	85.00	0.395434	0.382358	96.55	2.87	3239.93	81.16	92.60	21.84	55.30	22.94	56.02
30	65.00	0.629543	0.615919	97.54	5.46	12621.77	70.27	87.98	15.56	48.78	16.82	49.60

Table 7.7: (OLS, PermFlip, ϕ_{ME}, `Exchange`) gives the best results in terms of the numbers of cases where improvements are made.

n	Wins	err_{NN}	err_{FSS}	FSS/NN	Steps	FSSs	S_s	W_s	S_{rNN}	W_{rNN}	S_{rFSS}	W_{rFSS}
5	55.00	0.006750	0.001152	23.17	2.00	20.40	76.03	90.67	83.82	93.13	87.60	94.73
6	85.00	0.028788	0.004328	9.11	2.00	30.29	77.72	90.05	74.65	89.97	85.73	95.85
7	90.00	0.050661	0.025484	44.53	2.17	45.50	78.05	91.08	68.61	86.13	74.75	90.14
8	90.00	0.067445	0.048468	71.01	2.21	61.93	79.10	91.26	62.77	82.34	66.00	84.89
9	100.00	0.087002	0.067213	74.02	2.47	88.63	74.96	88.79	58.55	80.48	61.40	82.74
10	85.00	0.093545	0.074318	82.18	2.38	106.88	79.71	90.97	54.83	79.14	59.16	81.74
15	80.00	0.200556	0.168871	85.58	3.44	359.06	71.51	87.88	36.86	67.80	40.77	69.77
20	75.00	0.312128	0.270094	86.92	3.71	703.14	73.16	89.29	28.07	60.38	31.35	62.65
25	80.00	0.409966	0.352400	85.93	4.64	1389.00	65.23	83.03	21.79	54.89	24.57	57.44
30	90.00	0.634449	0.528474	85.31	5.39	2339.22	58.41	79.12	15.63	48.87	19.00	52.18

Table 7.8: If taking the significance of the improvement into account, we prefer (OLS, PermFliShift, ϕ_{ME}, `2-opt move`).

7.3.3 A geographic example

The tests on random distance matrices and random "flat" distance matrices have indicated that our local search strategy has some potential. We now illustrate the method on some "real life" data sets. Since the FSS can be represented by a planar split network, it is quite natural to test the approach with geographic distances. Here we pick ten European capital cities as shown in Figure 7.5 and check how well the FSS found by the local search is compared to the NeighborNet split system. Distances between the pairs of cities are computed from their longitudes and latitudes using the free City Distance Calculator provided by Geobytes Inc. (1). The distance matrix is shown in Table 7.9.

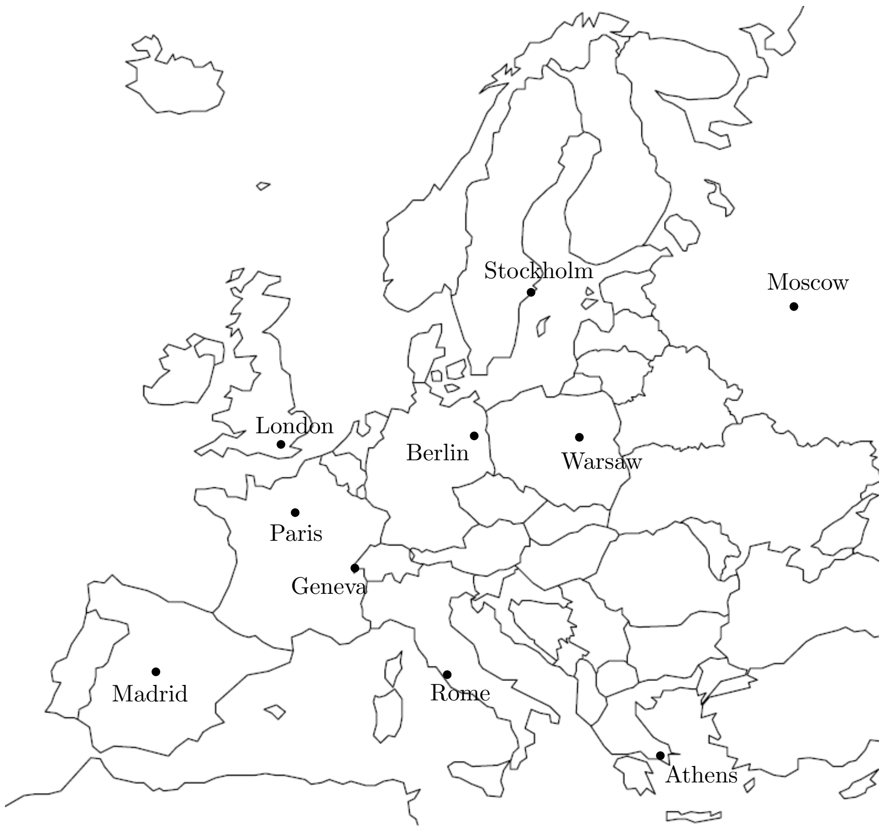


Figure 7.5: The 10 European capital cities on the map.

The split system found by NeighborNet has the NNLS error $\phi_{NNLS} \approx 21.84$, the network representing it is shown in Figure 7.6(a). The best FSS is found (OLS,

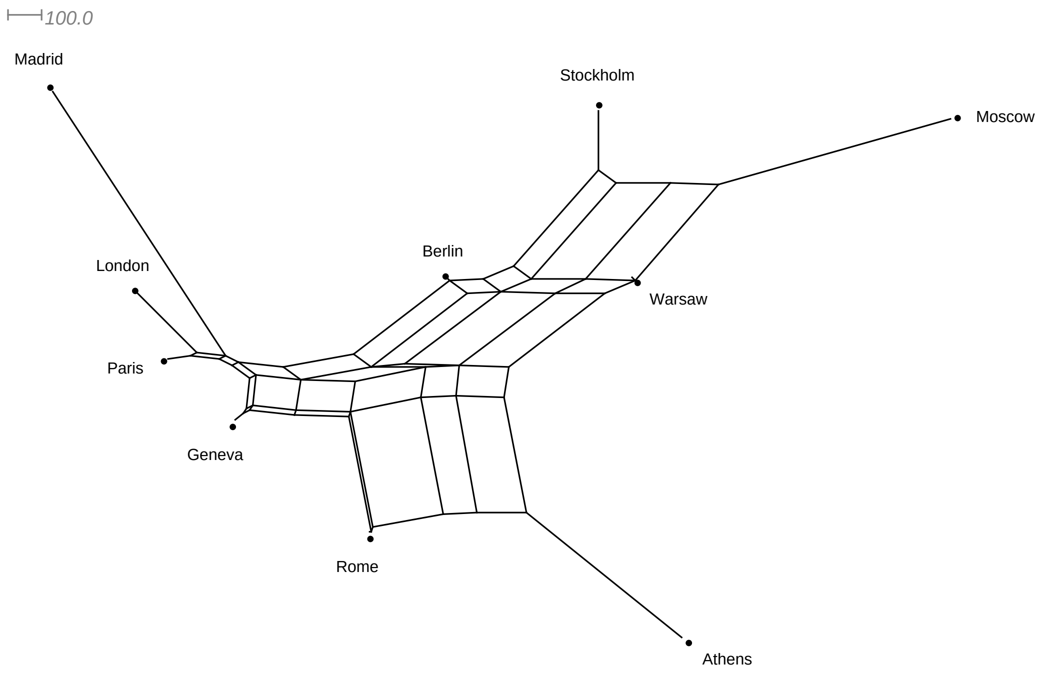
London	0									
Paris	343	0								
Madrid	1267	1054	0							
Rome	1434	1106	1363	0						
Geneva	748	411	1024	695	0					
Berlin	930	878	1870	1183	876	0				
Stockholm	1431	1545	2595	1978	1660	811	0			
Moscow	2498	2487	3441	2376	2416	1608	1227	0		
Warsaw	1447	1367	2292	1318	1267	516	809	1149	0	
Athens	2393	2098	2370	1053	1709	1804	2410	2231	1602	0

Table 7.9: The distance matrix between the capital cities.

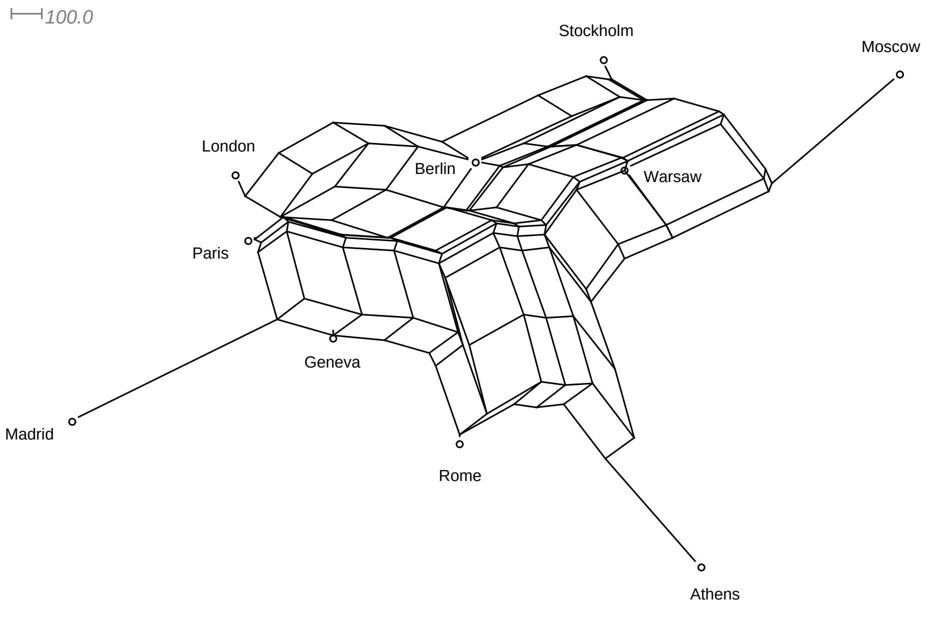
PermFlip, ϕ_{SumAbs}, `two-opt move`) and has $\phi_{NNLS} \approx 7.53$. The planar network representing the best FSS is shown in Figure 7.6(b). For both the networks we manually rearranged the vertices and edges so that the positions of the cities were as similar to those on the map as possible.

It is noticeable that Madrid is placed in a strange position. The problem is to get the city to the right point, the edge connecting the vertex to the rest of the network will have to cross some other edges. Besides this issue however, the NeighborNet network is not bad at all: The positions of the vertices pretty well represent the locations of the corresponding cities on the map.

In Figure 7.6(b), a major difference on the FSS network is that Berlin and Warsaw are placed “inside” the network and are separated from the boundary by splits of quite high weights. These positions resemble the geographic locations better than those in the NeighborNet network. Moreover, Madrid is well placed, too. Thus, at least visually, the FSS network represents the cities more appropriately than does the NeighborNet network. Quantitatively, the error of the FSS is under 35% that of the NeighborNet split system. All in all, the new approach appears to work better than NeighborNet in this example.



(a) The NeighborNet network, $\phi_{NNLS} \approx 21.84$.



(b) The best FSS network found by local search, $\phi_{NNLS} \approx 7.53$.

Figure 7.6: The "Capital 10" example, comparing the NeighborNet network to the best FSS network.

7.3.4 A biogeographic example

We now test the FSS approach on a published biological data set. We use the 19 nrITS sequences collected from 19 locations for the three *Ranunculus* species: *R. pachyrhizus*, *R. sericophyllus*, and *R. viridus* as mentioned in the paper by Winkworth *et. al.* (151). There are 20 sequences in the paper but location NE of Temple Basin of the first sequence (*sericophyllus*, CGTGGCCTAA) is not marked on the map so we omit this sequence. For this data set it is expected that the genetic data should be somewhat correlated with the geographical locations where the samples were collected from.

From the sequence data we were able to reconstruct the Split Decomposition network exactly as presented in Figure 1(e) in Winkworth *et. al.*'s paper. The network is pictured in Figure 7.7. We also use that data to build the NeighborNet network, shown in Figure 7.8(a); and from the distance matrix inferred from the sequences by NeighborNet, we find the best FSS possible with our local search, the network is shown in Figure 7.8(b).

Quantitatively, we see that both the NeighborNet split system and the FSS have much smaller errors than the Split Decomposition split system. However, the NeighborNet is slightly better than the FSS. The main difference between the NeighborNet and the Split Decomposition split networks is that the former parses the one node representing 8 taxa under Mt_Cook and the one node representing 3 taxa above Mt_Memphis in the latter into several nodes. Even though some of the new nodes are quite close together, they have fine tuned the distances and given better errors. A similar effect occurs when comparing the FSS and the Split Decomposition split system even though the parsing by the FSS is not as refined as the NeighborNet.

Split-wise (data not shown), both the NeighborNet split system and the FSS contain all 6 splits of the Split Decomposition system, all with high weights. NeighborNet and FSS share 9 splits, NeighborNet has 11 exclusive splits and FSS has 8. These numbers seem to agree with the hypothesis of "core" splits mention in Section 7.3.1.

With the same data set, we do a similar procedure on the distance matrix built on the geographic locations where the sequences are collected. Since the

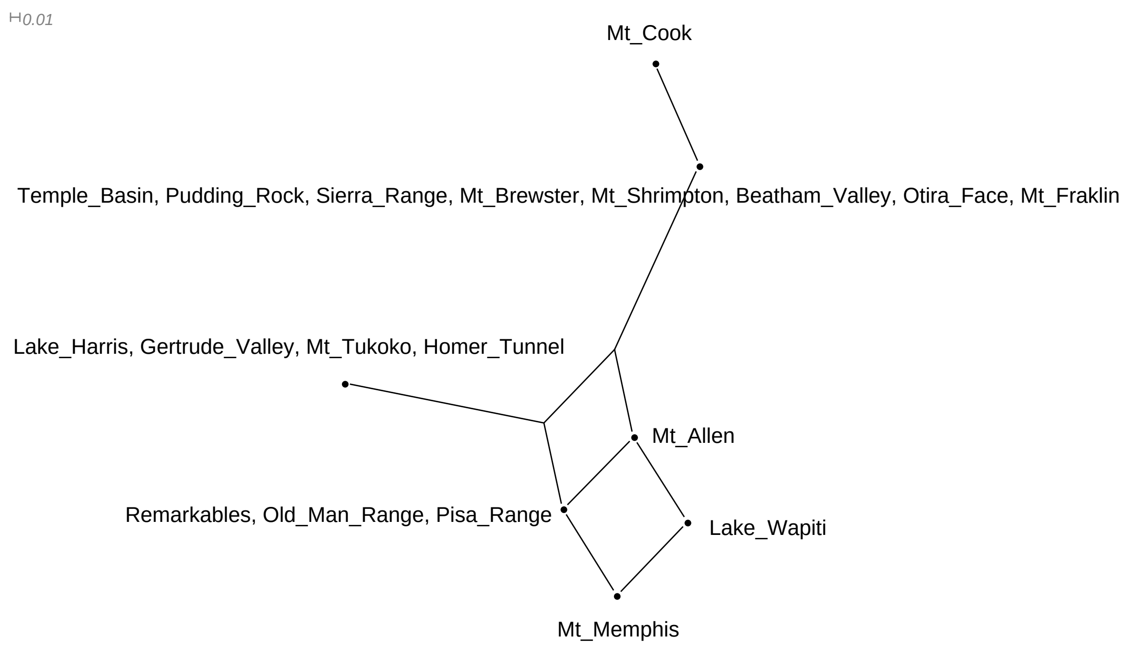
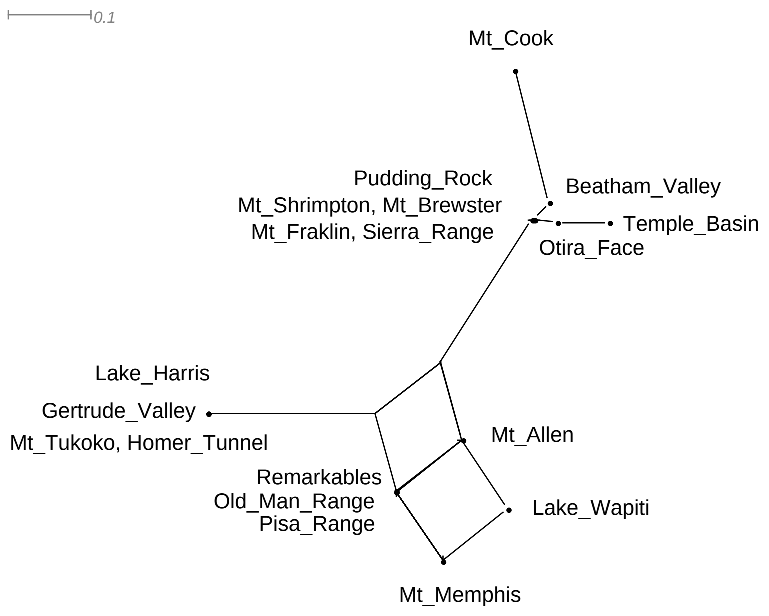
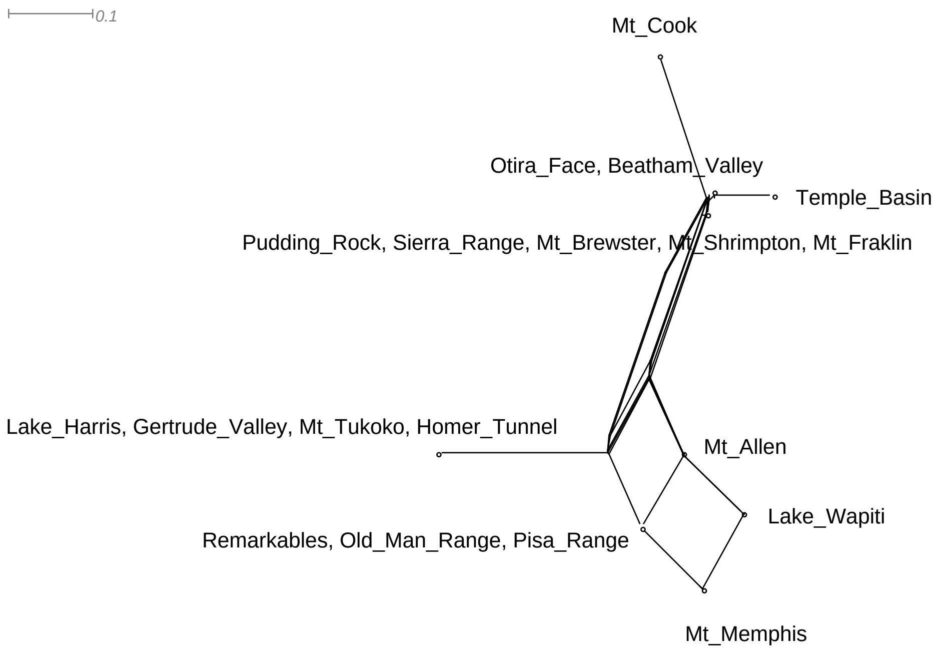

0.01
Mt_Cook
Temple_Basin, Pudding_Rock, Sierra_Range, Mt_Brewster, Mt_Shrimpton, Beatham_Valley, Otira_Face, Mt_Fraklin
Lake_Harris, Gertrude_Valley, Mt_Tukoko, Homer_Tunnel
Mt_Allen
Remarkables, Old_Man_Range, Pisa_Range
Lake_Wapiti
Mt_Memphis

Figure 7.7: The Split Decomposition network, $\phi_{NNLS} \approx 0.00381$, on the 19 *Ranunculus* sequences.



(a)

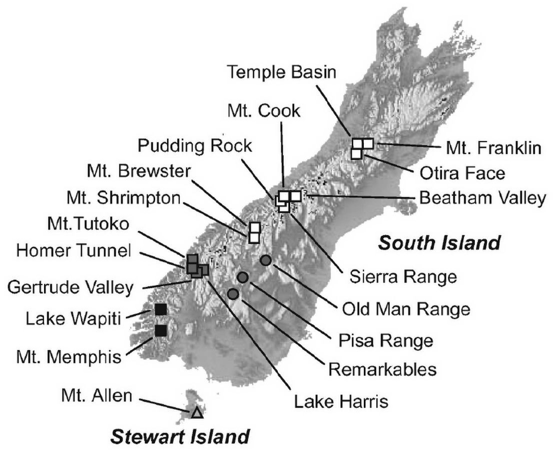

0.1
Mt_Cook
Otira_Face, Beatham_Valley
Temple_Basin
Pudding_Rock, Sierra_Range, Mt_Brewster, Mt_Shrimpton, Mt_Fraklin
Lake_Harris, Gertrude_Valley, Mt_Tukoko, Homer_Tunnel
Mt_Allen
Lake_Wapiti
Remarkables, Old_Man_Range, Pisa_Range
Mt_Memphis

(b)

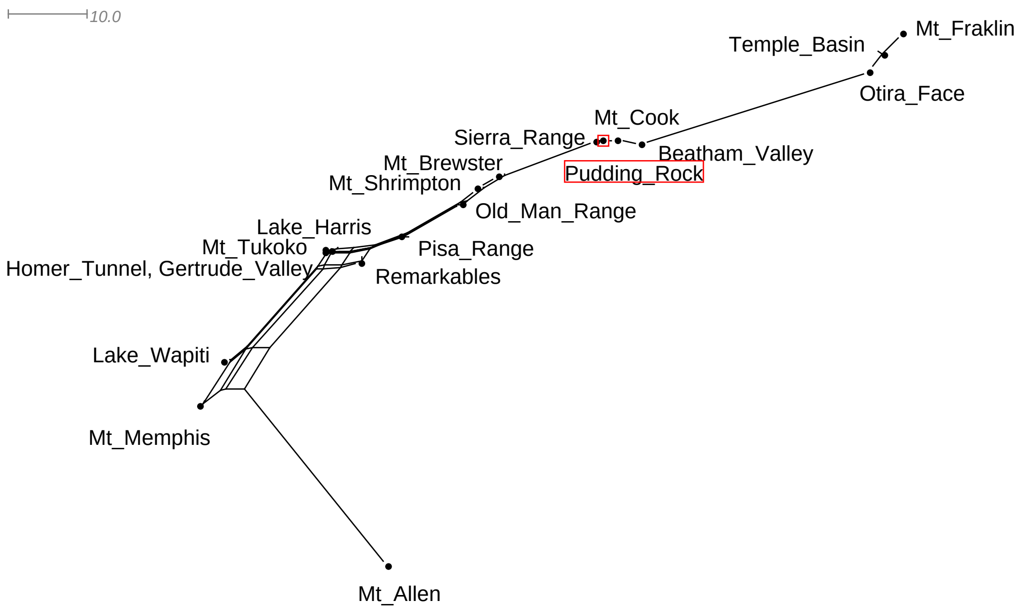
Figure 7.8: (a) The NeighborNet network, $\phi_{NNLS} \approx 0.00094$. (b) The best FSS network, found with (OLS, PermFlip, SumAbs, `2-opt move`), $\phi_{NNLS} \approx 0.00102$, built on the *Ranunculus* sequence data.

exact location data were not available, we approximated them by printing out a magnification of the sampling map (Figure 1(a) in (151), reproduced in Figure 7.9(a)), and using a ruler to measure the distances. The matrix is shown in Table 7.10; the Split Decomposition, NeighborNet and best FSS networks are shown in Figures 7.9(b), 7.10(a) and 7.10(b), respectively.

Visually all three networks seem to resemble the locations on the map quite well. However, when the errors are examined, NeighborNet and FSS are much better than Split Decomposition again. It is different this time though, in that the over all best is the FSS with an error of approximately 0.04975 compared to 0.05403 of NeighborNet and 0.78516 of Split Decomposition. The result should not be surprising since we have seen FSS works better than NeighborNet on "flat" distance matrices in Section 7.3.2. One can observe that on the FSS network, Lake_Harris is placed "inside", which seems to be reasonable due to its geographic location. By magnifying the network we can confirm that Siera_Range is also "inside" even though the splits separating it with the boundary have much smaller weights. The positions of these two taxa however result in the better error measure of the FSS.

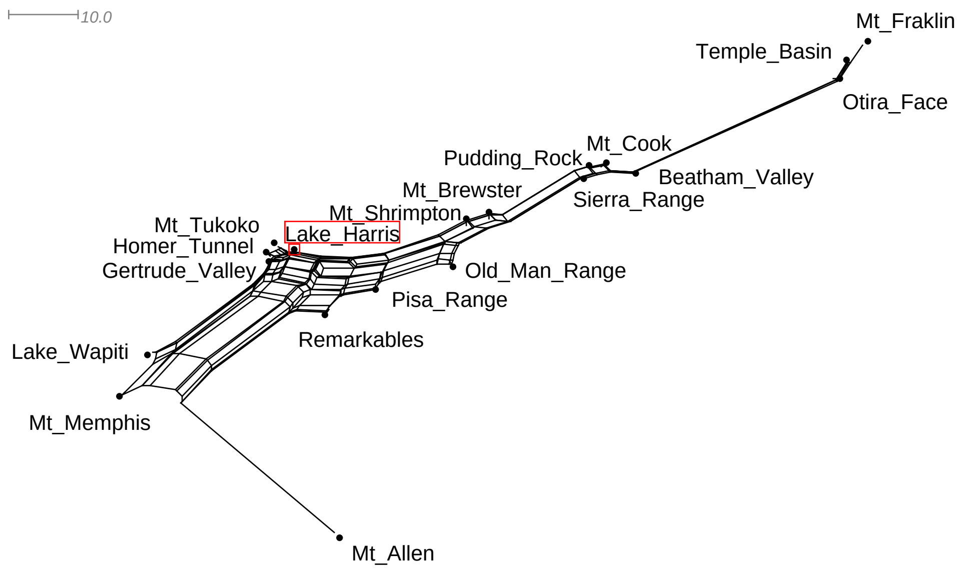


(a)

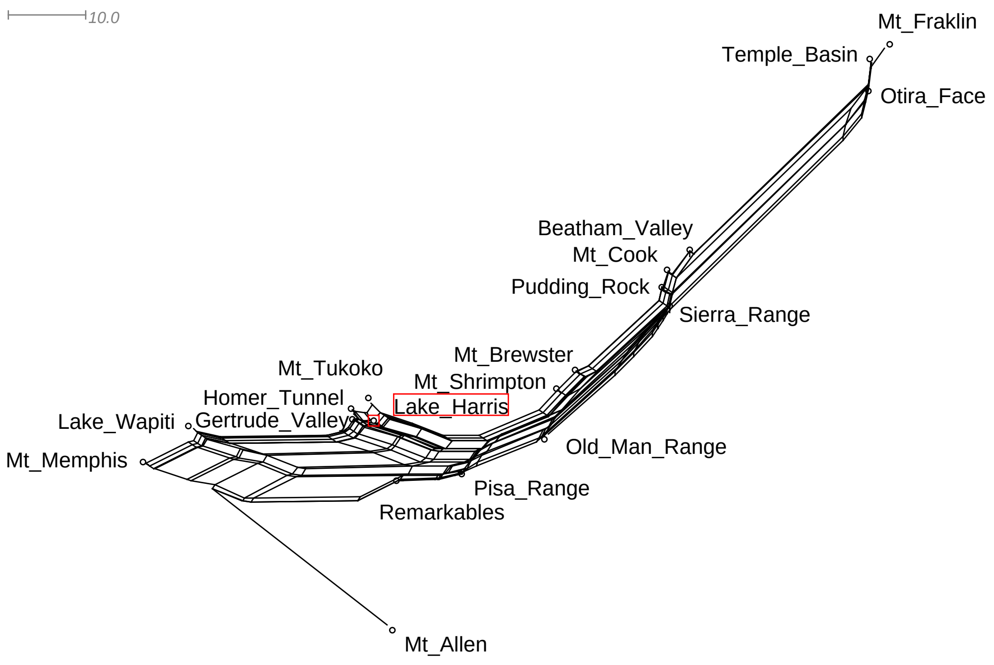


(b)

Figure 7.9: (a) The 19 sampling locations, except NE of Temple Basin, for the New Zealand *Ranunculus* (151). Reused with license (No. 2458431432805) from Oxford University Press. (b) The Split Decomposition network, $\phi_{NNLS} \approx 0.78516$, on the "geographic" distances between 19 sampling locations.



(a)



(b)

Figure 7.10: (a) The NeighborNet network, $\phi_{NNLS} \approx 0.05403$. (b) The best FSS network, found with (CLS, PermFlip, `Pair-Exchange`), $\phi_{NNLS} \approx 0.04957$, built on the "geographic" distances between 19 sampling locations.

Temple_Basin	0																		
Mt_Fraklin	4.9	0																	
Otira_Face	4.1	6.8	0																
Beatham_Valley	35.3	39.2	32.7	0															
Mt_Cook	39.8	43.6	37.2	5.4	0														
Pudding_Rock	42	46.1	39.7	7.1	2.7	0													
Sierra_Range	42.8	46.7	40	7.5	5	3.2	0												
Mt_Brewster	58.1	62.1	55.4	22.9	18.9	16.2	15.5	0											
Mt_Shrimpton	61.3	65.1	58.2	26	22.6	19.9	18.5	4.5	0										
Mt_Tukoko	88.9	92.8	86.2	53.7	49.2	46.6	46.2	31.1	29	0									
Homer_Tunnel	91.1	94.9	88.2	55.8	51.3	49.1	48.5	33	30.3	3.5	0								
Gertrude_Valley	90.8	94.7	88.1	55.7	51.2	48.8	48	32.5	29.8	6	2.8	0							
Lake_Harris	87.6	91.6	84.9	52.4	48.1	45.7	44.9	29.2	26.7	6.5	5	3.3	0						
Lake_Wapiti	113.2	117.2	110.7	78	73.9	71.4	70.5	55.1	52.2	26.1	23.2	22.5	25.9	0					
Mt_Memphis	119.9	123.5	116.8	84.5	80.8	78.1	77.1	61.9	58.6	34.4	31.1	30.5	32.9	10	0				
Old_Man_Range	65.5	68.6	62.1	31.5	29.5	27.5	24.6	15.3	11.5	32.5	32.5	31.2	27.1	51.2	56	0			
Pisa_Range	77.7	81.1	74.1	43.1	40.2	37.5	36	22.8	18.2	23.5	22.6	20.9	17.6	38.9	43.4	12.3	0		
Remarkables	86.1	89.4	82.8	51.6	48.9	46.5	44.7	31.1	26.8	23.2	21.7	18.5	16.9	32.9	36.1	20.6	8.5	0	
Mt_Allen	0.7	141.2	134.8	107.9	103.5	101.1	99	86.1	82	68	64.5	62.7	63.5	49	39.9	74.1	63.5	55	0

Table 7.10: The approximate geographic distance matrix between the 19 sampling locations of the *Ranunculus* data, distances are in millimeter.

Considering the splits in detail (data not shown), NeighborNet recovers 27 out of 28 splits produced by Split Decomposition and has 42 on its own. The corresponding numbers when comparing FSS with Split Decomposition are 27 and 33 - the FSS has less splits than does the NeighborNet. Nonetheless, NeighborNet and FSS share a significant number of 52 splits, leaving 17 for the former and 8 for the later. Again the idea of a group of “core” splits seems to be supported with this result.

Finally we will make a comparison between the DNA sequence-based split systems and the corresponding geographic split systems. This is motivated by the fact that to survive and develop, the species must adapt to the environment surrounding it, and the latter partly depends on its geographic location. Therefore the species’ DNA may display some geographic “signals” and vice versa, which we hope might be detectable in terms of shared splits.

With Split Decomposition (data not shown), 3 out of 6 splits of the genetic split system are shared by the geographic split system who has 28 splits. The corresponding numbers with NeighborNet are 7, 20, 69; and those with the best FSSs are 6, 17, 60. Over all three pairs of split systems, about 30% - 50% splits of the genetic split system are “recovered” with geographic data. This is a good indication that the species have adapted according to geography to some degree.

However, a closer examination reveals that with Split Decomposition (data not shown), the shared splits contribute to 48.61% total weight of the genetic but only 4.12% total weight of the geographic split systems. The corresponding numbers with NeighborNet are 63.64% and 6.34%; and those with the best FSSs are 66.06% and 5.67%. It seems that the splits which display important genetic characteristics are not that highly valued geographically. A possible explanation for the observation is that in this data set, the species’ genes might be more influenced by some other environmental/biological factors rather than by the geographic locations alone. A thorough study on all the other factors may help clarify the hypothesis but it is not in the scope of this thesis.

7.4 Concluding remarks

We have proposed a framework employing a number of parameters to search for a weighted FSS to represent a given distance matrix. The framework is a simple local search model but the parameters are independent and could be employed in other search algorithms, hopefully to produce improved results.

We have tested the new approach on both random and real life data. Initial results are positive: In many cases the split system found by the FSS approach accommodates the distance matrix better than does NeighborNet. The new method works particularly well when the input matrix came from a FSS. The test also suggest that given a distance matrix, there can be more than one split system that accommodates it well, and there seems to be a group of "core" splits that appear in all or most such split systems. In the final chapter we will discuss some possible future directions.

Chapter 8

Conclusions and future work

8.1 Conclusions

Phylogenetic diversity is an important measure used to evaluate the evolutionary diversity of a set of species. Depending on specific data and context, there can be different ways to define PD and thus, different approaches to study the problems concerning this measure. In this research, based on splits, we have developed novel methods to solve the PD optimisation problems in various situations.

In Chapter 2, we gave a mathematical background to the thesis and introduced MPD as a main problem to be considered. Chapter 3 dealt with the complexity aspects of MPD: We proved that MPD itself and related optimisation problems, namely $WAPD$, MED, and MO, for all of which the PD measure proposed by Faith (47) is adopted, are NP-hard. We then considered similar optimisation problems with alternative measures of diversity, namely the number of segregating sites (in an alignment of gene sequences) and the AD measure, and proved that they are NP-hard too.

Algorithmically, an NP-hard problem can have polynomial time solutions under some special types of input data. In Chapter 4, we proposed efficient algorithms for two situations. First, by exploiting the structure of phylogenetic trees and incorporating the idea for computing tree cores, we developed an $O(n)$ algorithm for the MPD problem on trees. This solution is more efficient than those that had been published before and is obviously optimal. The tree-like structure was then used in a dynamic programming framework to derive a polynomial time algorithm for the optimisation problem based on the AD measure on the tree.

Next, we studied the MPD problem for more general split systems, namely affine split systems and circular split systems in Chapter 5. For affine split systems, by applying some geometric analyses we derived an $O(kn^3)$ algorithm. Up to that time, $O(kn^3)$ was also the complexity of the best algorithm for the MPD problem on circular split systems, which are a subset of affine split systems. By making use of geometric considerations we developed an $O(kn + n \log n)$ algorithm for this problem, which to our best knowledge is the one with the best complexity to date.

In Chapters 6 and 7 we considered a new theme: Constructing FSSs from distance matrices. The work is motivated by the fact that geographic locations where biological data are collected may leave some signals in the species' genetic features, which we hope to detect.

In Chapter 6, we formally defined FSSs, shown that they are a generalisation of the affine split systems, and indicated how to construct a planar split network to represent a FSS. We also defined several operations to manipulate the FSSs. These operations were used in Chapter 7 as the basis to search for a weighted FSS to represent a distance matrix. This FSS-based approach proved to work better than NeighborNet in many cases, especially if the matrices are induced from geographic locations. It also suggested an interesting idea that a group of "core" splits may appear in all, or most, FSSs that represent the same distance matrix. Finally, the result on the biogeographical data seemed is in some agreement with the hypothesis about the correlation between geographic and genetic data.

8.2 Future work

8.2.1 Counting the number of full FSSs

A basic, but maybe not so easy problem concerning FSSs is to count the number of full FSSs of a fixed set of taxa. We know that each p-sequence gives a full FSS. The number of p-sequences can be calculated through the number of permutations and the number of sequences of swapping points. Given a set of n taxa, the number of permutations is $n!$. The number of sequences of swapping points, denoted as $\mathcal{A}_n$, can be computed using Stanley's formula (137)

$$\mathcal{A}_n = \frac{\binom{n}{2}!}{(2n-3) \times (2n-5)^2 \times (2n-7)^3 \times \cdots \times 5^{n-3} \times 3^{n-2}}.$$

Let $\mathcal{B}_n$ be the number of p-sequences, we have $\mathcal{B}_n = n!\mathcal{A}_n$. This quantity increases extremely quickly with n as illustrated in Table 8.1.

n	$n!$	$\mathcal{A}_n$	$\mathcal{B}_n$
2	2	1	2
3	6	2	12
4	24	16	384
5	120	768	92,160
6	720	292,864	210,862,080
7	5,040	1,100,742,656	5,547,742,986,240

Table 8.1: How the number of p-sequences increases with the increase of the number of taxa.

However, it is evident that more than one p-sequence may yield the same FSS. Operations `Swapping` and `Shifting` are good examples illustrating this fact. As a consequence, finding formal criteria to check if two p-sequences derive the same FSS is important, at least mathematically. These criteria could then be used to form a rule for counting the number of full FSSs. Work in this direction may also be helpful to understand the space of FSSs and help drive the local search more efficiently.

8.2.2 The MPD problem on FSSs

We have studied problem *MPD* in several special cases in Chapters 3, 4 and 5. It is natural to extend the questions to FSSs. In particular we have the following problem:

MaximumPhylogeneticDiversityOnFlatSplitSystems (F-MPD)
Input: a set X with n elements

a full flat split system $\mathfrak{S} = \mathfrak{S}(\mathcal{P})$
where $\mathcal{P} = (\pi^0, P)$, π^0 is a permutation of elements of X
and P is a sequence of swapping points
a split weight function $\omega : \mathfrak{S} \to \mathbb{R}_{>0}$
an integer $k \in \{1, \ldots, n\}$

Output: a subset $Y \subseteq X$ with the property that $|Y| = k$ and
$PD(Y) = \max\{PD(W)|W \subseteq X, |W| = k\}$

An obvious algorithmic question is “Is F-MPD NP-hard?” If the answer to this question is “No”, it is natural to develop efficient algorithms for F-MPD. In Chapter 5 by making use of the special structure of the split system and some geometry we have given an $O(kn^3)$ algorithm for affine split systems, which are very closely related to FSSs. The special structure of wiring diagram of the FSS might be of use to develop an efficient algorithm for problem F-MPD: The selection of a taxon is now a selection of a wire, and each combination of the wires gives rise to a set of intersections which are splits with assigned weights. A dynamic programming approach may therefore be a good starting to explore the problem.

If the answer to the question is “Yes”, finding approximation methods for F-MPD might be necessary. Remember that the greedy approach worked well on trees for finding an optimal solution to the MPD problem, so it might be sensible to explore further how well greedy might perform for FSSs. One could start with two wires whose intersection has the biggest weight, then keep adding wires so that the contribution of weights assigned to the additional intersections is maximal. An extensive evaluation of the greedy approach would be useful even in the case where a polynomial algorithm for F-MPD exists, since such algorithm may bee too expensive to be applicable on very large data sets.

Besides, to be more applicable, the F-MPD problem could be studied with additional constraints added. The extra condition may be geographical, biological (110), economical (18) or ecological (51). It would be natural to study problem F-MPD with such additional requirements.

8.2.3 Improving the search approach for FSSs

There are several directions that could be followed to improve the search for a weighted FSS to represent a given distance matrix. The local search that we currently employed needs to be sped up further to be practical. At the moment it can find results in a reasonable time for data sets of up to around 30 taxa with the CLS approach and a bit more with the OLS approach. We have identified that the most and second most expensive parts of the search process are to fit a distance matrix to a FSS using the Lawson and Hanson's NNLS algorithm and to do the same job using the Gaussian elimination. We have also observed that (data not shown) the improvements of the errors between consecutive neighborhoods are usually limited. For this reason, instead of considering all split systems in the neighborhood to choose the best one, we could maybe start with a new set of neighbors every time a reduction on the error is made.

Another way to enhance the search might be to select another starting point for the search instead of always starting with the NeighborNet split system. One can easily start with a random or a number of random FSSs, then select the best output. The initial split system could also be selected using the Multidimensional Scaling (MDS) approach (34): Input the distance matrix into a MDS procedure to get the corresponding set of points, generate an affine split system from these points, and then use this split system as the input for the search. Varying the input FSS would help explore the search space more extensively and avoid the trap of local optima. In addition it could help testing the hypothesis for the "core" splits that we mentioned above.

Due to the cost of the CLS approach, in practice the OLS approach is much more likely to be useful. In that case choosing a suitable error measure for the weight vector that may contain negative values is important. Our tests have shown that among the three measures ϕ_{ME}, ϕ_{SumAbs} and ϕ_{SumPos}, the last measure, even though seems to be quite correlated to the second one, which yields good solutions for several cases, always gives worst solutions. Therefore trying out other weight vector-based measures could be worth exploring.

Note that in our search we have employed only 3 out of 5 permuting strategies presented in Section 6.4.4, leaving `Reversal` and `Prune & Regraft` untouch. Incorporating those strategies into the search could be reasonable and might help improve the search.

Independently on the choice of starting FSS for the search, the weight vector-based error measure for the OLS approach, and the permuting strategy, one could design other search frameworks using other heuristic methods such as genetic algorithm (65; 102) or simulated annealing (27; 70; 90). For genetic algorithms, each FSS could be considered as a pair of "genes", one for the initial permutation and one for the sequence of swapping points. Having them in the form of sequences of numbers already, it would be convenient to define genetic operations such as cross-over and mutation for these "genes". The use of simulated annealing with the possibility of accepting a solution with a worse object function score could help avoid getting trapped in local optima.

Last but not least, whichever search strategy is employed, it seems worthy to find a way to incorporate the tabu search method (64) into the framework to reduce the number of FSSs on which the input distance matrix is to be fitted. Recall that different p-sequences may yeild the same FSS, and solving the WCF problem is computationally expensive, we can maintain a list of FSSs that have already been considered so that each unique FSS is fitted with the input distance matrix only once.

8.3 Final remarks

Split systems are a popular tool to study the evolutionary history of species. They are used to represent the evolutionary relationships of different complexity, with or without conflicting signals. Consequently, they are the object for theoretical and practical optimisation problems in phylogenetics. In the last two decades a lot of work has been put into the field with the number of publications increasing year by year.

Based on splits, this thesis has shed new light onto the field of phylogenetics by studying several important optimisation problems. The systematic investigation into FSSs has produced some interesting results and has much promise. It has also laid the basis for and raised some open questions for further research, which we look forward to seeing studied in the future.

Bibliography

[1] Geobytes city distance calculator. http://www.geobytes.com/CityDistanceTool.htm. 129

[2] E. Adams. Consensus techniques and the comparison of taxonomic trees. *Systems Zoology*, **21**:390–397, 1971. 18

[3] E. Adams. N-trees as nestings: Complexity, similarity and consensus. *Journal of Classification*, **3**:299–317, 1986. 18

[4] P. K. Agarwal and M. Sharir. Pseudo-line arrangements: Duality, algorithms, and applications. *SIAM Journal on Computing*, **34**:526–552, 2005. 81

[5] A. Aggarwal, M. M. Klawe, S. Moran, P. Shor, and R. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, **2**:195–208, 1987. 64, 68

[6] B.L. Allen and M. Steel. Subtree transfer operations and their induced metrics on evolutionary trees. *Annals of Combinatorics*, **5**(1):1–15, 2001. 15, 100

[7] R. Ariew. *Ockham's Razor: A historical and philosophical analysis of Ockham's principle of parsimony*. PhD thesis, University of Illinois, 1976. 14

[8] K. Atteson. The performance of the neighbor-joining methods of phylogenetic reconstruction. *Algorithmica*, **25**:251–278, 1999. 17

[9] H. J. Bandelt and A. W. M. Dress. A canonical decomposition theory for metrics on a finite set. *Advances in Mathematics*, **92**:47–105, 1992. 19, 20, 22

[10] G. M. Barker. Phylogenetic diversity: A quantitative framework for measurement of priority and achievement in biodiversity conservation. *Biological Journal of the Linnean Society*, **76**(2):165–194, 2002. 24

[11] P. H. Barrett, P. J. Gautrey, S. Herbert, D. Kohn, and S. Smith, editors. *Charles Darwins Notebooks, 1836-1844: Geology, Transmutation of Species, Metaphysical Enquiries*. Cambridge University Press, 1987. 5

[12] C. Berge. *Théorie des graphes et ses applications*. Collection Universitaire de Mathématiques. Dunod, Paris, 1958. 6

[13] O. R. P. Bininda-Emonds. *Bioconsensus*, chapter MRP supertree construction in the consensus setting, pages 231–242. Number 61 in DIMACS. American Mathematical Society, 2003. 18

[14] O. R. P. Bininda-Emonds, J. L. Gittleman, and M. A. Steel. The (super)tree of life: Procedures, problems, and prospects. *Annual Reviews in Ecology and Systematics*, **33**:265–289, 2002. 18

[15] A. Bjorner, M. L. Vergnas, B. Sturmfels, N. White, and G. M. Ziegler. *Oriented Matroids*, **46** of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, 2006. 74, 77, 97, 98

[16] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and Robert E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, **7**:448–461, 1973. 44

[17] M. Bordewich, O. Gascuel, K. T. Huber, and V. Moulton. Consistency of topological moves based on the balanced minimum evolution principle of phylogenetic inference. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, **6**(1):110–117, 2009. 100

[18] M. Bordewich and C. Semple. Nature reserve selection problem: A tight approximation algorithm. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **5**:275–280, 2008. 35, 144

[19] M. Bordewich, C. Semple, and A. Spillner. Optimizing phylogenetic diversity across two trees. *Applied Mathematics Letters*, **22**(5):638–641, 2009. 35

[20] J. E. Boyce, D. P. Dobkin, R. L. Drysdale, and L. J. Guibas. Finding extremal polygons. *SIAM Journal on Computing*, **14**:134–147, 1985. 62, 64, 65, 70

[21] D. Bryant. On the uniqueness of the selection criterion in neighbor-Joining. *Journal of Classification*, **22**(1):3–15, June 2005. 17

[22] D. Bryant and A. Dress. Linearly independent split systems. *European Journal of Combinatorics*, **28**:1814–1831, 2007. 60, 79, 106, 107

[23] D. Bryant and V. Moulton. NeighborNet: An agglomerative method for the construction of phylogenetic networks. *Molecular Biology and Evolution*, **21**(2):255–265, 2003. 20, 23

[24] D. Bryant, V. Moulton, and A. Spillner. Computing planar split graphs. In *The Annual New Zealand Phylogenetics Meeting*, 2007. 59

[25] D. Bryant, V. Moulton, and A. Spillner. Consistency of the Neighbor-Net algorithm. *Algorithms for Molecular Biology*, **2**(8), 2007. 23

[26] P. Buneman. The recovery of trees from measures of dissimilarity. In D. Kendall and P. Tautu, editors, *Mathematics in Archaeological and Historical Sciences*, pages 387–395. Edinburg University Press, 1971. 13

[27] V. Cerny. A thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, **45**:41–51, 1985. 146

[28] B. Chandra and M. Halldórsson. Approximation algorithms for dispersion problems. *Journal of Algorithms*, **38**:438–465, 2001. 41

[29] B. Chandra, H. Karloff, and C. Tovey. New results on the old k-opt algorithm for the TSP. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1994. 101

[30] R. Chandrasekaran and A. Daughety. Location on tree networks: p-centre and n-dispersion problems. *Mathematics of Operations Research*, **6**:50–57, 1981. 41

[31] V. Chepoi and B. Fichet. A note on circular decomposable metrics. *Geometriae Dedicata*, **69**:237–240, 1998. 64

[32] N. Christofides and S. Eilon. Algorithms for large-scale travelling salesman problems. *Operational Research Quarterly*, **23**(4):511–518, 1972. 101

[33] H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2001. 26, 30, 31, 32, 52, 54, 56

[34] T. F. Cox and M. A. A. Cox. *Multidimensional Scaling*. Monographs on Statistics & Applied Probability. Chapman and Hall/CRC, 2000. 145

[35] A. Czygrinow. Maximum dispersion problem in dense graphs. *Operations Research Letters*, **27**:223–227, 2000. 41

[36] R. Desper and O. Gascuel. *Mathematics of Evolution and Phylogeny*, chapter The minimum evolution distance-based approach to phylogenetic inference. Oxford University Press, 2005. 105

[37] R. Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer-Verlag, Heidelberg, 2005. 6

[38] E. Dominguez and J. Munoz. A neural model for the p-median problem. *Computers and Operations Research*, **35**(2):404–416, 2008. 41

[39] A. W.M. Dress and D. H. Huson. Constructing splits graphs. *IEEE Transactions on Computational Biology and Bioinformatics*, **1**(3), 2004. 21, 23

[40] Z. Drezner. The p-centre problem - heuristic and optimal algorithms. *Journal of the Operational Research Society*, **35**:741–748, 1984. 41

[41] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. EATCS - Monographs in Theoretical Computer Science. Springer, July 1987. 60, 84

[42] A. W. F. Edwards and L. L. Cavalli-Sfroza. The reconstruction of evolution. *Annals of Human Genetics*, **27**:104–105, 1963. 14

[43] B.C. Emerson, S. Forgie, S. Goodacre, and P. Oromi. Testing phylogeographic predictions on an active volcanic island: *Brachyderes rugatus* (Coleoptera: Curculionidae) on La Palma (Canary Islands). *Molecular Ecology*, **15**:449–458, 2006. 40

[44] D. Eppstein, M. H. Overmars, G. Rote, and G. J. Woeginger. Finding minimum area k-gons. *Discrete and Computational Geometry*, **7**:45–58, 1992. 59, 62, 63

[45] J. Evans, L. Sheneman, and J. Foster. Relaxed Neighbor Joining: A fast distance-based phylogenetic tree construction method. *Journal of Molecular Evolution*, **62**:785–792, 2006. 18

[46] D. Faith. Phylogeny and conservation. *Systematic Biology*, **56**:690–694, 2007. 24

[47] D. P. Faith. Conservation evaluation and phylogenetic diversity. *Biological Conservation*, **61**:1–10, 1992. 24, 34, 44, 141

[48] D. P. Faith. The role of the phylogenetic diversity measure, PD, in bio-informatics: Getting the definition right. *Evolutionary Bioinformatics Online*, **2**:301–307, 2006. `http://la-press.com/cr_data/files/f_EBO-2-Faith-2_171.pdf`. 24

[49] D. P. Faith. *Conservation Biology: Evolution in Action*, chapter Phylogenetic diversity and conservation. Oxford University Press, New York, 2008. 24

[50] D. P. Faith and A. Baker. Phylogenetic diversity (PD) and biodiversity conservation: Some bioinformatics challenges. *Evolutionary Bioinformatics Online*, 2006. http://la-press.com/journals.php?pa=abstract&content_id=148. 24

[51] B. Faller, C. Semple, and D. Welsh. Optimizing phylogenetic diversity with ecological constraints. *Annals of Combinatorics*, 2010. In press. 144

[52] J. Felsenstein. Evolutionary trees from DNA sequences: A maximum likelihood approach. *Journal of Molecular Evolution*, **17**:368–376, 1981. 14

[53] J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, 2004. 12, 15, 16, 19

[54] S. Felsner. *Geometric Graphs and Arrangements: Some Chapters from Combinatorial Geometry*. Vieweg Verlag, 2004. 74, 76, 77, 81

[55] W. Fitch. Toward defining the course of evolution: Minimal change for a specific tree topology. *Systematic Zoology*, **20**:406–416, 1971. 15

[56] W. M. Fitch and E. Margoliash. Construction of phylogenetic trees. *Science*, **155**:279–284, 1967. 16

[57] F. Forest, R. Grenyer, M. Rouget, T. J. Davies, R. M. Cowling, D. P. Faith, A. Balmford, J. C. Manning, S. Proches, M. v d Bank, G. Reeves, T. A. J. Hedderson, and V. Savolainen. Preserving the evolutionary potential of floras in biodiversity hotspots. *Nature*, **445**:757–760, February 2007. 24

[58] L. R. Foulds and R. L. Graham. The Steiner problem in phylogeny is NP-complete. *Advances in Applied Mathematics*, **3**:43–49, 1982. 15

[59] R. D. Galvao. Uncapacitated facility location problems: Contributions. *Pesquisa Operacional*, **24**(1):7–38, April 2004. 41

[60] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Series of Books in the Mathematical Sciences. W. H. Freeman, 1979. 30, 32, 35, 37, 38

[61] O. Gascuel, D. Bryant, and F. Denis. Strengths and limitations of the minimum-evolution principle. *Systematic Biology*, **50**(5):621–627, 2001. 108

[62] O. Gascuel and M. Steel. Neighbor-Joining revealed. *Molecular Biology and Evolution*, **23**(11):1997–2000, 2006. 18

[63] J. B. Ghosh. Computational aspects of the maximum diversity problem. *Operations Research Letters*, **19**:175–181, 1996. 41

[64] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997. 146

[65] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Kluwer Academic Publishers, 1989. 146

[66] D. Goldstein, A. Ruíz-Linares, M. Feldman, and L. L. Cavalli-Sforza. An evaluation of genetic distances for use with microsatellite loci. *Genetics*, **139**:463–471, 1995. 16

[67] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The John Hopkins University Press, 3^{rd} edition, 1996. 107

[68] J. E. Goodman and R. Pollack. *New trends in discrete and computational geometry*, **10** of *Algorithms and Combinatorics*, chapter Allowable sequences and order types in discrete and computational geometry, pages 103–134. Springer-Verlag, Berlin, 1993. 74

[69] J.E. Goodman and R. Pollack. On the combinatorial classification of nondegenerate configurations in the plane. *Journal of Combinatorial Theory*, **29**(2):220–235, 1980. 74

[70] V. Granville, M. Krivanek, and J. P. Rasson. Simulated annealing: A proof of convergence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **16**(6):652–656, 1994. 146

[71] E. Haeckel. *Generelle Morphologie der Organismen, II.* George Reiner, Berlin, 1866. xi, 5, 8

[72] R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, **26**(2):147–160, 1950. 16

[73] J. A. Hartigan. Minimum mutation fits to a given tree. *Biometrics*, **29**:53–65, 1973. 15

[74] K. Helsgaun. General k-opt submoves for the Lin-Kernighan TSP heuristic. *Mathematical Programming Computation*, **1**:119–163, 2009. 101

[75] M. D. Hendy and D. Penny. Branch and bound algorithms to determine minimal evolutionary trees. *Mathematical Biosciences*, **60**:133–142, 1982. 15

[76] G. Hickey, M. Blanchette, P. Carmi, A. Maheshwari, and N. Zeh. An approximation algorithm for the Noahs ark problem with random feature loss. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2010. PrePrint. 35

[77] D. M. Hillis, C. Moritz, and B. K. Mable. *Molecular Systematics*. Sinauer Associates, Inc., 1996. 108

[78] D. Hochbaum. Approximating covering and packing problems: set cover, vertex cover, independent set, and related problems. In D. Hochbaum, editor, *Approximation algorithms for NP-hard problems*. PWS Publishing, 1997. 34

[79] B. R. Holland. *Evolutionary analyses of large data sets: Trees and beyond*. PhD thesis, Massey University, 2001. 37

[80] K. T. Huber and V. Moulton. *Mathematics of Evolution and Phylogeny*, chapter Phylogenetic networks, pages 178–204. Oxford University Press, 2005. 19, 23

[81] K. T. Huber, V. Moulton, P. Lockhart, and A. Dress. Pruned median networks: A technique for reducing the complexity of median networks. *Molecular Phylogenetics and Evolution*, **19**:302–310, 2001. 23

[82] J. P. Huelsenbeck, J. P. Bollback, and A. M. Levine. Inferring the root of a phylogenetic tree. *Systematic Biology*, **51**(1):32–43, January 2002. 12

[83] D. H. Huson. ISMB-Tutorial: Introduction to phylogenetic networks. `www-ab.informatik.uni-tuebingen.de/research/phylonets/ismb2005.pdf`. 19

[84] D. H. Huson and D. Bryant. Application of phylogenetic networks in evolutionary studies. *Molecular Biology and Evolution*, **23**(2):254–267, February 2006. xi, xiii, 19, 20, 24, 104

[85] D. H. Huson, T. Dezulian, T. Klopper, and M. A. Steel. Phylogenetic super-networks from partial trees. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **1**(4):151 – 158, 2004. 23

[86] N. C. Jones. *An introduction to bioinformatics algorithms (Computational Molecular Biology)*. MIT Press, 2004. 98

[87] T.H. Jukes and C.R. Cantor. *Evolution of Protein Molecules*. Academic Press, New York, 1969. 18

[88] K. Kalmanson. Edgeconvex circuits and the travelling salesman problem. *Canadian Journal of Mathematics*, **27**:1000–1010, 1975. 64

[89] K. K. Kidd and L. A. Sgaramella-Zonta. Phylogenetic analysis: Concepts and methods. *American Journal of Human Genetics*, **23**(3):235–252, 1971. 108

[90] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, **220**(4598):671–680, 1983. 146

[91] I. J. Kitching, P. L. Forey, C. J. Humphries D. M., and Williams. *Cladistics: The Theory and Practice of Parsimony Analysis*. Number 11 in The Systematics Association Publication. Oxford University Press, 1998. 14, 16

[92] J. Krarup and P. Pruzan. The simple plant location problem: Survey and synthesis. *European Journal of Operations Research*, **12**(1):36–81, 1983. 41

[93] K. N. Laland, J. Odling-Smee, and M. W. Feldman. Niche construction, biological evolution, and cultural change. *Behavioral and Brain Sciences*, **23**:131–146, 2000. 18

[94] C. L. Lawson and B. J. Hanson. *Solving Least Squares Problems*. Prentice-Hall, Englewood Cliffs, NJ, 1974. 107

[95] F. Levi. Die teilung der projektiven ebene durch gerade oder pseudogerade. *Berichte, Mathematische-Physikalische Klasse, Akademie der Wissenschaften Leipzig*, **78**:256–267, 1926. 81

[96] D. Levy and L. Pachter. The Neighbor-Net algorithm. 2007. 105

[97] L. Lewis and P. Lewis. Unearthing the molecular phylodiversity of desert soil green algae (chlorophyta). *Systematic Biology*, **54**:936–947, 2005. 36

[98] P. J. Lockhart, P.A. McLenachan, D. Havell, D. Glenny, D. H. Huson, and U. Jensen. Phylogeny, dispersal and radiation of New Zealand alpine buttercups: Molecular evidence under split decomposition. *Annals of the Missouri Botanical Garden*, **88**:458–477, 2001. 23

[99] K-T. Mak and A. J. Morton. Distances between traveling salesman tours. *Discrete Applied Mathematics*, **58**(3):281–291, 1995. 101

[100] V. Makarenkov, D. Kevorkov, and P. Legendre. *Bioinformatics*, **6** of *Applied Mycology and Biotechnology*, chapter Phylogenetic network construction approaches, pages 61–97. Elsevier, 2006. 19, 23

[101] B. Mau and M. Newton. Phylogenetic inference for binary data on dendrograms using Markov chain Monte Carlo. *Journal of Computational and Graphical Statistics*, **6**:122–131, 1997. 19

[102] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1999. 146

[103] C. D. Michener and R. R. Sokal. A quantitative approach to a problem in classification. *Evolution*, **11**:130–162, 1957. 16

[104] M. J. Milne and L. J. Milne. Evolutionary trends in caddis worm cases construction. *Annals of the Entomological Society of America*, **32**:533–542, 1966. 6

[105] E. Minch, A. Ruz-Linares, and D. Goldstein. Genetic distance, 2008. `http://hpgl.stanford.edu/projects/microsat/distance.html`, URL accessed July 26, 2010. 16

[106] B. Q. Minh, S. Klaere, and A. von Haeseler. Phylogenetic diversity within seconds. *Systematic Biology*, **55**:769–773, 2006. 24, 34, 43, 71

[107] B. Q. Minh, S. Klaere, and A. von Haeseler. Phylogenetic diversity on split networks. unpublished manuscript, 2007. 26, 64, 71

[108] B. Q. Minh, F. Pardi, S. Klaere, and A. v. Haeseler. Budgeted phylogenetic diversity on circular split systems. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **6**(1):22–29, 2009. 36

[109] G.W. Moore, M. Goodman, and J. Barnabas. An iterative approach from the standpoint of the additive hypothesis to the dendrogram problem posed by molecular data sets. *Journal of Theoretical Biology*, **38**:423–457, 1973. 15

[110] V. Moulton, C. Semple, and M. A. Steel. Optimizing phylogenetic diversity under constraints. *Journal of Theoretical Biology*, **246**(1):186–194, May 2007. 24, 35, 37, 38, 144

[111] L. Nakhleh, T. Warnow, C.R. Linder, and K. St. John. Reconstructing reticulate evolution in species - Theory and practice. *Journal of Computational Biology*, **12**(6-7):796–811, 2005. 23

[112] K. Nehring and C. Puppe. Modelling phylogenetic diversity. *Resource and Energy Economics*, **26**(2):205–235, June 2004. 24

[113] M. Nei. Estimation of average heterozygosity and genetic distance from a small number of individuals. *Genetics*, **89**:583–590, 1978. 16

[114] M. Nei and S. Kuma. *Molecular Evolution and Phylogenetics*. Oxford University Press, New York, 2000. 16

[115] M. Newman. Simple models of evolution and extinction. *Computing in Science and Engineering*, **2**(1):80–86, Jan/Feb 2000. 18

[116] F. Pardi and N. Goldman. Species choice for comparative genomics: Being greedy works. *PLoS Genetics*, **1**(6), 2005. 24, 43, 44

[117] F. Pardi and N. Goldman. Resource-aware taxon selection for maximizing phylogenetic diversity. *Systematic Biology*, **56**(3):431–44, 2007. 36

[118] O. A. Prokopyev, N. Kong, and D. L. Martinez-Torres. Discrete optimization - the equitable dispersion problem. *European Journal of Operational Research*, 2008. In press. 41

[119] B. Raphael. Code for Gaussian elimination. `http://imacwww.epfl.ch/Team/Raphael/BookWiley2003/java-illustrations/GaussianElimination/GaussianElimination.java.html`. 111

[120] S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, **42**:299–310, 1994. 41

[121] G. Ringel. Teilungen der ebene durch geraden oder topologische geraden. *Mathematische Zeitschrift*, **64**(1):79–102, 1956. 97

[122] G. Ringel. Uber geraden in allgemeiner lage. *Elemente der Mathematische*, **12**:75–82, 1957. 97

[123] D. F. Robinson. Comparison of labelled trees with valency three. *Journal of Combinatorial Theory*, **11**:105–119, 1971. 15

[124] A. S. L. Rodrigues and K. J. Gaston. Maximising phylogenetic diversity in the selection of networks of conservation areas. *Biological Conservation*, **105**:103–111, 2002. 24, 35, 36

[125] A. S. Romer. *Vertebrate Paleontology.* University of Chicago Press, 3^{rd} edition, 1966. 6

[126] A. Rzhetsky and M. Nei. Theoretical foundation of the minimum-evolution method of phylogenetic inference. *Molecular Biology and Evolution*, **10**(5):1073–1095, 1993. 16, 108

[127] N. Saitou and M. Nei. The Neighbor-Joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, **4**(4):406–25, 1987. 16

[128] M. J. Sanderson, A. Purvis, and C. Henze. Phylogenetic supertrees: Assembling the trees of life. *Trends in Ecology & Evolution*, **13**:105–109, 1998. 18

[129] E. Schroder. Vier combinatorische probleme [Four combinatorial problems]. *Zeitschrift fur Maththematik und Physik*, **15**:361–376, 1870. 115

[130] H. Schwöbbermeyer and J. T. Kim. A comparative analysis of biodiversity measures. In *ECAL '99: Proceedings of the 5th European Conference on Advances in Artificial Life*, pages 119–128, London, UK, 1999. Springer-Verlag. 36

[131] C. Semple and M. A. Steel. A supertree method for rooted trees. *Discrete Applied Mathematics*, **105**:147–158, 2000. 18

[132] L. Sheneman, J. Evans, and J. A. Foster. Clearcut: A fast implementation of relaxed Neighbor Joining. *Bioinformatics*, **22**(22):2823–2824, 2006. 18

[133] A. Shioura and T. Uno. A linear time algorithm for finding a k-tree core. *Journal of Algorithms*, **23**:281–290, 1997. 43

[134] M. Slatkin. A measure of population subdivision based on microsatellite allele frequencies. *Genetics*, **139**:457–462, 1995. 16

[135] A. Spillner and V. Moulton. Drawing flat split systems. (in preparation). 84

[136] A. Spillner, B. T. Nguyen, and V. Moulton. Computing phylogenetic diversity for split systems. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **5**(2):235 – 244, April-June 2008. 30, 43, 59

[137] R.P. Stanley. On the number of reduced decompositions of elements of Coxeter groups. *European Journal of Combinatorics*, **359-372**, 1984. 143

[138] M. A. Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of Classification*, **9**:91–116, 1992. 18

[139] M. A. Steel. Phylogenetic diversity and the greedy algorithm. *Systematic Biology*, **54**(4):527–529, 2005. 24, 40, 43, 44

[140] M. A. Steel and C. Semple. *Phylogenetics*. Oxford University Press, 2003. 10, 12, 15, 100, 115

[141] M. Sun. Solving the uncapacitated facility location problem using tabu search. *Computers and Operations Research*, **33**(9):2563–2589, 2006. 41

[142] D.L. Swofford and G.J. Olsen. *Molecular Systematics*, chapter Phylogeny reconstruction, pages 411–501. Sinauer Associates, 1990. 15, 100

[143] F. Tajima. Evolutionary relationship of DNA in finite populations. *Genetics*, **105**:437–460, 1983. 40

[144] A. Tamir. Algorithmic results on facility location problems on networks. In *EWI 2007 Book*, 2007. 41

[145] R. Uehara and Yushi Uno. Efficient algorithms for the longest path problem. In *Proceeding of International Symposium on Algorithms and Computation (ISAAC)*, 2004. 44

[146] G. A. Waterson. On the number of segregating sites in genetical models without recombination. *Theoretical Population Biology*, **7**:256–276, 1975. 39

[147] M. L. Weitzman. The Noah's ark problem. *Econometrica*, **66**(6):1279–1298, 1998. 35

[148] E. Welzl. Constructing the visibility graph for n-line segments in $O(n^2)$ time. *Information Processing Letters*, **20**(44):167–171, 1985. 84

[149] R. Wetzel. *Zur Visualisierung abstrakter Ähnlichkeitsbeziehungen*. PhD thesis, Fakultät Mathematik, Universität Bielefeld, Bielefeld, Germany, 1995. 22

[150] P.H. Williams, C. J. Humphries, and R. I. Vane-Wright. Measuring biodiversity: Taxonomic relatedness for conservation priorities. *Australian Systematic Botany*, **4**:665–679, 1991. 24

[151] R. C. Winkworth, D. Bryant, P. J. Lockhart, D. Havell, and V. Moulton. Biogeographic interpretation of splits graphs: Least squares optimization of branch lengths. *Systematic Biology*, **54**(1):56–65, 2005. xiv, 132, 135, 136

[152] Z. Yang and B. Rannala. Bayesian phylogenetic inference using DNA sequences: A Markov chain Monte Carlo method. *Molecular Biology and Evolution*, **14**:717–724, 1997. 14, 19